Proyecto final de carrera

Project Harvester

Plataforma web de proyectos colaborativos

Título	Project Harvester – Web de proyectos colaborativos
Autor	Juan María Martín Espinosa
Fecha	23 de junio de 2014
Directora	Ana Edelmira Pasarella Sánchez
Departamento	Lenguajes y Sistemas Informáticos (LSI)
Titulación	Ingeniería Informática técnica de gestión
Centro	Facultat d'Informàtica de Barcelona (FIB)
Universidad	Universitat Politècnica de Catalunya (UPC) Barcelona Tech

Motivación

La idea que originó desarrollar esta plataforma surgió de mi afición a tocar el piano. Esto llegó a oídos de un profesor que en su tiempo libre tocaba música en solitario pero que deseaba formar un grupo y me planteó unirme. Aunque al final no llegó a formarse el grupo, la idea de que había que conseguir alguna forma de reunir personas con algún objetivo en común de forma sencilla y facilitando la comunicación entre ellos, estuvo en mi mente por algún tiempo. Luego de comentar esta inquietud con mi actual directora de proyectos Edelmira Pasarella, mi idea tomó forma y se planteó desde la perspectiva de proyectos que necesitan colaboración para desarrollarse y acabar con éxito, trayendo consigo un beneficio ya sea económico o social.

Adicionalmente, pensé que el desarrollo de una plataforma de este tipo me brindaría la oportunidad de aplicar todo lo aprendido durante la carrera, siendo el desarrollo web una disciplina que engloba muchos campos y áreas. Además, llevar a cabo este proyecto me planteó un reto debido a mi poca experiencia previa en el desarrollo de plataformas web 2.0 y sus lenguajes de programación. Esto último despertó mi interés, ya que una premisa importante que he aprendido a lo largo de la carrera, es que es de gran importancia tener la capacidad de adquirir nuevos conocimientos y saber buscar por uno mismo las herramientas necesarias para obtenerlos.

Índice

1. Introducción

Tras varios años de crisis económica, para muchos se hace vital agudizar el ingenio para salir adelante. Es necesario, por tanto, la existencia de herramientas que faciliten compartir nuevas ideas. La colaboración de personas de diferentes perfiles pero con intereses comunes es muy importante - muchas veces vital - para transformar y llevar a término proyectos que podrían llegar a ser irrealizables, debido a su complejidad o la falta de recursos.

Aunque existen varias plataformas que ofrecen financiación colectiva para los proyectos, repositorios de ideas o entornos que promueven debates colectivos, hasta donde hemos visto, no existe en la actualidad plataforma alguna que mezcle todas esas características y las aproveche para llegar a convertir las ideas en proyectos financiados y desarrollados con éxito.

En este marco de referencia hemos creado una plataforma 2.0 para facilitar el intercambio de ideas y/o propósitos comunes para llevar a cabo proyectos ya sean de carácter social o iniciativa privada. En particular, nuestra plataforma facilitará la creación de proyectos(crowdsourcing moderado) y la obtención de recursos(*crowdfunding* dinámico) para realizarlos. En efecto, esta plataforma ofrece un punto de encuentro donde personas que no necesariamente se conocen o tienen algún vínculo previamente puedan reunirse y aportar sus conocimientos y destrezas para así darle forma a meras ideas hasta convertirlas en proyectos con unos objetivos concretos, factibles de ser desarrollados y con un mecanismo de financiación colectiva.

Esta memoria está organizada como sigue. En la Sección 2, presentamos el objetivo general y los objetivos específicos del proyecto así como las restricciones para su realización. En la Sección 3, describimos las especificación de la plataforma, realizando primeramente un análisis y comparativa entre otras soluciones. Con este análisis estableceremos las fortalezas y debilidades de plataformas conocidas a fin de robustecer nuestra propuesta y presentar una opción novedosa con respecto a lo que podemos encontrar hoy en día. Posteriormente, son descritas las entidades, casos de uso y funcionalidades de la plataforma. En la Sección 4, explicaremos como ha sido desarrollada la plataforma. En la Sección 5, esbozamos el diseño principal de la interfaz del usuario. En la Sección 6, realizamos una descripción de los recursos necesarios para el proyecto, tales como el tiempo y costes tanto estimados como reales. En la penúltima sección presentamos las conclusiones a las que hemos llegado una vez terminado el proyecto. Finalmente, planteamos algunas de las extensiones a desarrollar en el futuro para la plataforma.

2. Objetivos del proyecto

2.1. Objetivo general del proyecto

El objetivo principal es crear una plataforma que aproveche el potencial que brinda el entorno web para facilitar el intercambio de ideas y/o propósitos comunes para llevar a cabo un proyecto ya sea de carácter social o iniciativa privada. Además, permitir reunirse para conocer distintos puntos de vista y negociarlos, para la realización de una idea de la forma más fácil, eficiente y ordenada.

Implementar cada proyecto basado en una entidad grupo de forma que haya control de quiénes participan en éste, creando así el entorno necesario para concentrar la comunicación e interacción. De esta forma pueden asignarse tareas y plazos a cada participante del proyecto permitiendo la comunicación entre ellos de forma eficiente y rápida.

Facilitar el crecimiento y consecución de proyectos proporcionando tanto, diferentes vías de comunicación entre los usuarios como una vía de financiación. Las vías de comunicación dentro de cada proyecto a implementar serán un chat y un muro de mensajes. Proporcionar con esto y otras herramientas un entorno privado de forma que pueda ser gestionado con facilidad quién participará activamente en la consecución de este.

Habilitar un sistema de financiación crowdfunding incentivado con recompensas para que sea utilizado siempre que sea necesario durante el desarrollo de cada proyecto publicado.

Con el objetivo de facilitar también la comunicación entre cualquier usuario de la plataforma crear un sistema de correo interno entre ellos, además de un sistema de marcaje a los usuarios tanto para destacarlos como para bloquearlos. Además de permitir crear perfiles de usuario con un mínimo de información suficiente como para generar la confianza necesaria sobre los participantes de cada proyecto al resto de usuarios.

Sobretodo crear un entorno seguro y fiable para el crecimiento y desarrollo de cualquier tipo de proyecto.

2.2. Objetivos específicos del proyecto

Hemos desglosado los objetivos para el desarrollo del proyecto en varias tareas. La primera de todas es realizar una investigación preliminar para determinar las características y funcionalidades principales de las que estará compuesta la plataforma. Para ello, un punto importante es la comparativa con otras plataformas y así determinar los puntos fuertes y débiles de estas y nuestra propuesta.

Una vez llegado al consenso de las necesidades de la plataforma, elegir un marco de trabajo que nos facilite el desarrollo de esta.

Analizar gestores de contenidos o CMS debido a que no solo proporcionan la gestión de usuarios y contenido sino además también aspectos no menos importantes como la seguridad, gestión de bases de datos, etc. para escoger el que mejor encaje con las necesidades del proyecto.

Una vez escogido el CMS adecuado, realizar el proceso de aprendizaje de este por la carencia previa de conocimientos sobre el uso de estos.

En paralelo, otro objetivo es delimitar que entidades compondrán la plataforma además de todas sus funcionalidades y necesidades externas, tales como el servidor donde alojarla y tecnologías complementarias al CMS que extiendan las posibilidades de este para conseguir cumplir los objetivos.

Una vez acabada la etapa anterior viene el desarrollar la plataforma, y su interfaz de usuario. Durante el desarrollo es necesario comprobar el correcto funcionamiento de las funcionalidades implementadas y sobretodo que encajen con las ya implementadas. Además de comprobar que tenga el funcionamiento esperado en un entorno con usuarios reales, en este caso, en el servidor escogido.

Durante el desarrollo del proyecto es necesario tener actualizada la documentación que nos va a servir de guía para el correcto desarrollo.

Por último, describir posibles futuras funcionalidades a implementar para mejorar la experiencia y las posibilidades de la plataforma.

2.3. Contextualización/Restricciones

Legislativa

- El gobierno está impulsando una nueva ley para regular el crowdfunding en España. Esta obliga a tener un capital inicial para el propietario o propietarios de la plataforma y restricciones en las contribuciones por proyecto y plataforma que aun están por determinar.

- Según la legislación vigente todo usuario que se registre en una red social, como esta plataforma, deberá tener al menos 14 años de edad.

- Tendremos en cuenta la legislación vigente para crowdfunding además del tratamiento de datos privados, incluyendo las cookies.

Tecnológica

- El servidor escogido para la plataforma además de económico debe permitir la instalación de aplicaciones ajenas a éste, tales como NodeJS o apacheSolr. Además tendrá que suministrar una capacidad de memoria suficiente para la carga de trabajo de la plataforma al menos en sus inicios.

Tiempo

- La plataforma tendría que estar desarrollada en no más de año y medio, teniendo en cuenta no solo el desarrollo e implementación de ésta, sino además la documentación e investigación previa relacionada con esta.

Usabilidad

- El diseño de la plataforma tiene que tener un diseño atractivo para los usuarios y que de confianza al usuario para utilizar la plataforma y todas sus opciones, incluyendo la financiación de proyectos.

- Aplicaremos técnicas SEO para mejorar el posicionamiento de la plataforma en los motores de búsqueda de Internet además de facilitar la accesibilidad.

3. Especificación

En esta sección presentamos la especificación de la plataforma. Comenzamos por hacer un estudio comparativo de la diferentes soluciones que se ofrecen actualmente en Internet. A continuación describimos las entidades que componen la plataforma y luego, presentamos la especificación UML: diagramas de clases, casos de uso y funcionalidades principales.

3.1. Análisis y comparativa con otras plataformas

Durante el desarrollo de este proyecto hemos intentado crear algo distinto y que tenga una marcada diferencia con otras soluciones, para ello han sido muchas las plataformas web que han sido visitadas para este propósito. A continuación presentamos un resumen de las conclusiones.

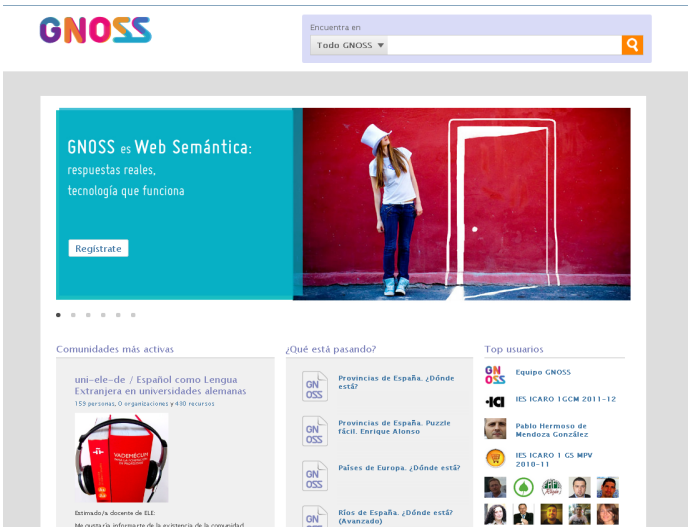
3.1.1. Repositorios de ideas

ideas4all.com. Es un repositorio de ideas con el cuál compartir tu idea o reto con el resto de personas. Dispone de muchas opciones para la búsqueda de ideas, tanto nube de tags, categorías. Además cada idea tiene asociada un sencillo foro público en forma de comentarios para hacer crecer la idea. Pero no dispone de grupo privado con la posibilidad de seleccionar quien participa, y por tanto, tampoco de vías de comunicación exclusivas de grupo o eventos, ni tampoco financiación. El inconveniente principal de esta plataforma es que el debate colectivo producido para cada idea propuesta, aparentemente, no deja entrever si llega a llevarse a término.

3.1.2. Web semántica

Las webs semánticas son una opción muy útil a la hora de buscar recursos ya que su sistema de etiquetado avanzando de contenido permite una búsqueda muy exacta de lo que se necesita encontrar. Este sistema facilitaría mucho el encuentro de proyectos donde poder participar.

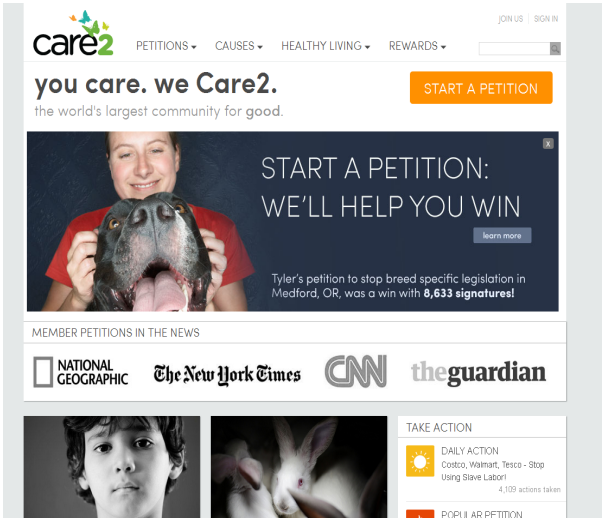
gnoss.com: Esta web semántica permite etiquetar documentos, webs y otros recursos para así encontrarlos de forma rápida y precisa. Podría servir para publicar proyectos ya que su sistema de etiquetado proporciona una búsqueda fiable de contenidos. Aunque no implementa un filtrado por campos como lo sería en nuestra plataforma, por descripción, categorías, candidatos, etc. De la misma forma que en ideas4all.com cada recurso tiene asociado un pequeño foro de comentarios. Además ofrece la posibilidad de crear espacios de debate entre otros recursos. Pero tal como otras soluciones carece de grupo privado, por tanto de gestión de usuarios por cada recurso, vías de comunicación internas y financiación.



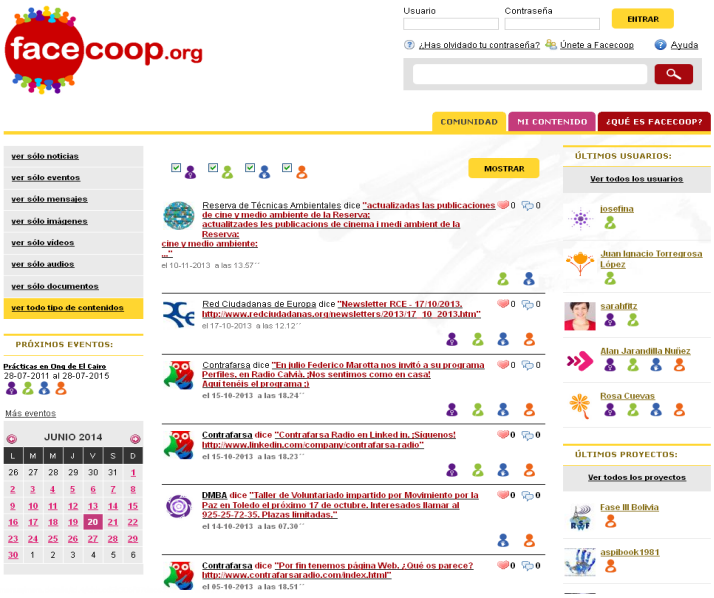
3.1.3. Cooperación social

Una de las características de la plataforma que planteamos es la posibilidad de crear proyectos de tipo social.

Plataformas como **care2.com** o **change.org** permiten crear peticiones para emprender acciones para conseguir una meta social. El gran éxito sobretodo del segundo, viene dado por las facilidades que ofrecen para firmar las peticiones, también para difundir las peticiones a través de las redes sociales y sobretodo son peticiones que tienen llegan a tener éxito. Sin embargo, la carencia de grupos privados con lo que conlleva impide que puedan llevarse a cabo proyectos o acciones sociales en colaboración real más allá de evitar acciones o actos perjudiciales.



facecoop.org: En cambio si que ofrece la posibilidad de crear grupos privados, aunque solo con un sencillo muro de mensajes público. Además, es posible subir recursos como documentos, vídeos e imágenes y crear eventos pero no específicos del grupo sino para toda la plataforma. Como ventaja permite clasificar los grupos o proyectos por más de una categoría, aunque solo hay 4 disponibles. Pero no permite especificar en un sitio visible y aparte candidatos necesarios para llevar a cabo proyectos ni la financiación, ni otros medios de comunicación que no sea el muro de mensajes.

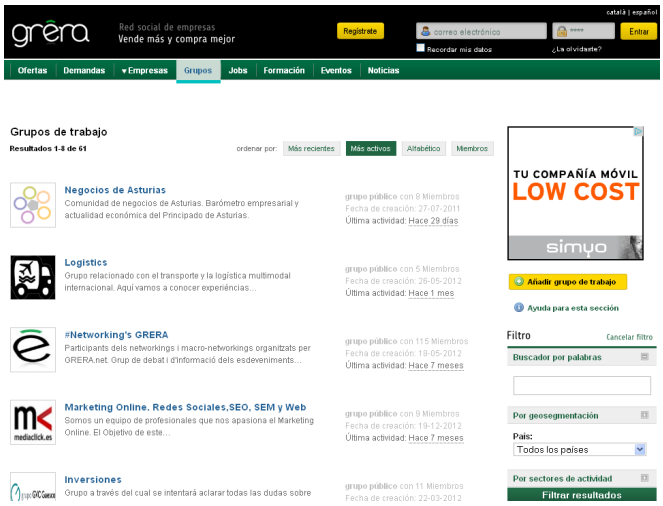


3.1.4. Emprendedores

Prácticamente todos las soluciones analizadas en esta sección ofrecen las mismas características aunque centradas en compartir conocimiento y ofrecer espacios de debate, más que en el desarrollo de proyectos concretos. Por tanto, en todas existe una sección de foro de debates.

redsocial.emprendego.com, **lacancilleria.com** y **emprendedores20.ning.com** son redes sociales muy similares enfocadas a emprendedores. Son bastante completas. Permiten crear grupos tanto abiertos como cerrados con muro de mensajes. Además, para todos los usuarios de las plataformas hay un chat, (no disponible en lacancilleria.com), foro de debates, blog, etc. pero ninguno de estos de forma cerrada para cada grupo. Tampoco permiten la financiación ya que son webs para crear espacios de debate y compartir conocimiento.

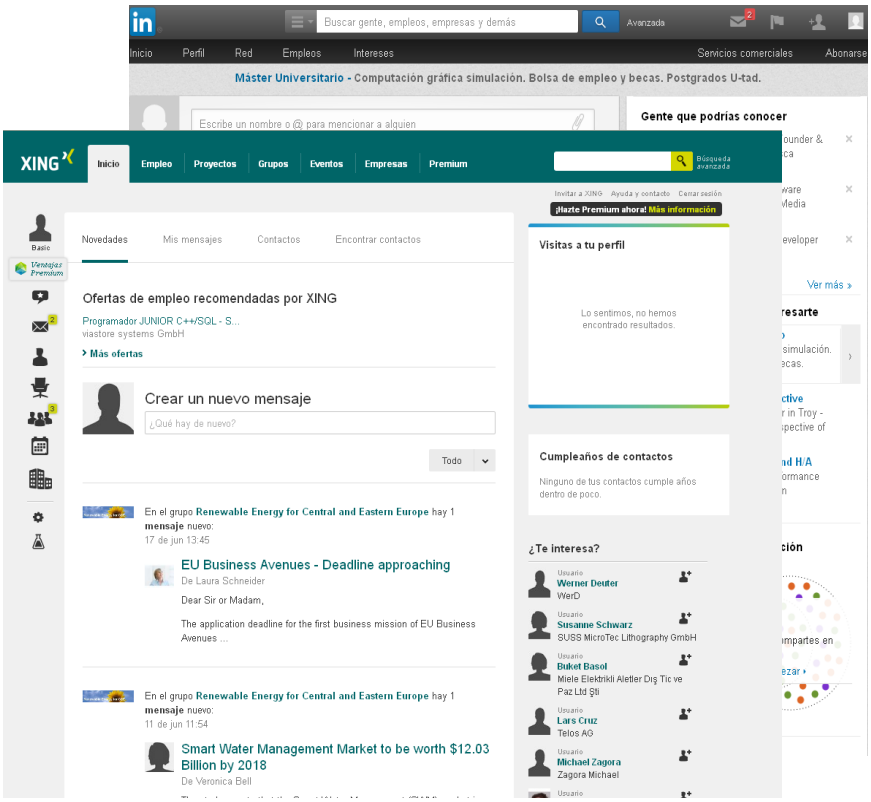
grera.net guarda cierto parecido con los anteriores pero sin foro de debate, ni chat. El inconveniente principal es que está centrado en empresas ya creadas, por tanto, no apta para alguien con una nueva idea a desarrollar que no tenga ya una empresa creada. Además, es utilizada sobretodo para ofrecer y pedir servicios o trabajos, además de compartir conocimiento.



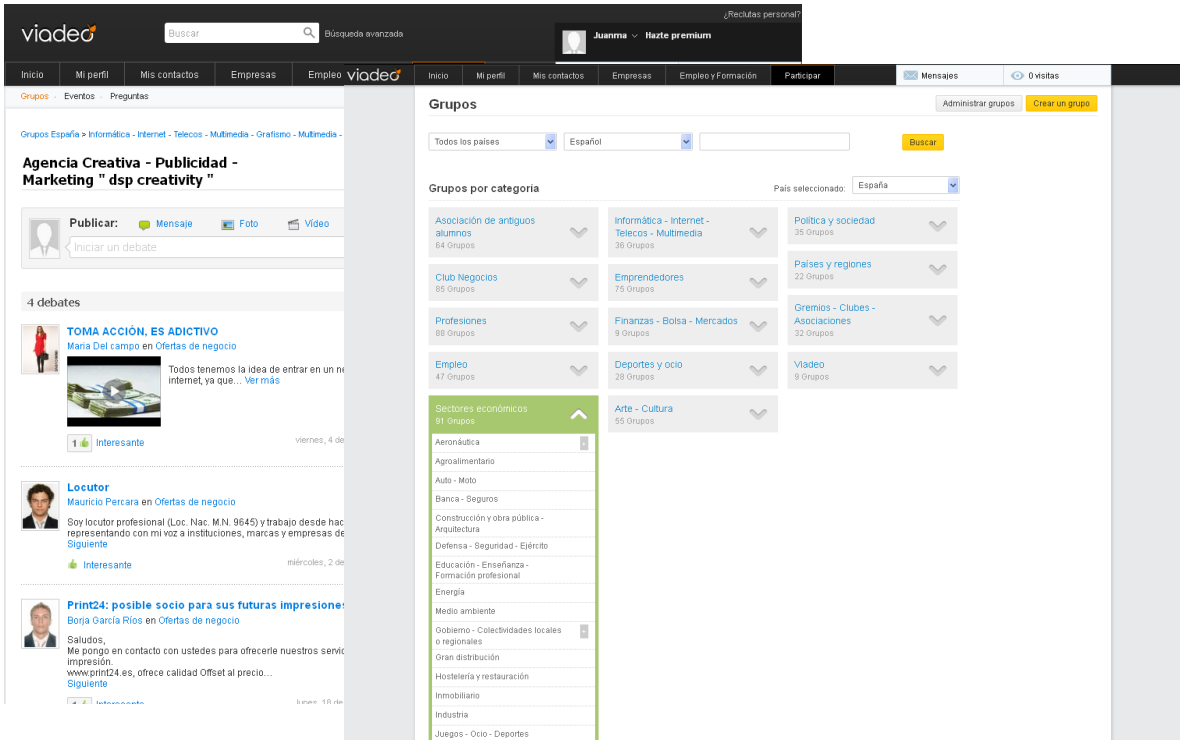
econred.es en cambio solo dispone de grupos para ofrecer espacios de debate y formación online.

redemprendeverde.es está centrado solo publicar emprendedores del ámbito de la ecología y proyectos ya realizados con éxito.

Xing y Linkedin son dos redes sociales para profesionales y emprendedores que cuentan con muchos usuarios. Tienen características muy similares, no son solo un directorio de empresas y empleos sino que también permite que los usuarios establezcan contactos entre ellos. Además posibilitan la creación de grupos tanto públicos como privados. Aunque ninguno de los dos permite la financiación, Xing tiene un apartado diferenciado para los proyectos, donde es posible enviar una solicitud para entrar. Un inconveniente de ambos es que no tienen la posibilidad de crear eventos de grupo, aunque si públicos, ni tampoco chat. La gran diferencia entre los dos es el sistema de búsqueda, siendo el de Xing más avanzando ya que permite buscar por zona, por tags y por temática.



Viadeo es una red social con mucho parecido a otras tales como Xing. Marcando la diferencia con una gran cantidad de posibilidades a la hora de categorizar los grupos, además de permitir eventos por grupo y compartir otros recursos como fotos y vídeos.



3.1.5. Webs de búsqueda de empleo

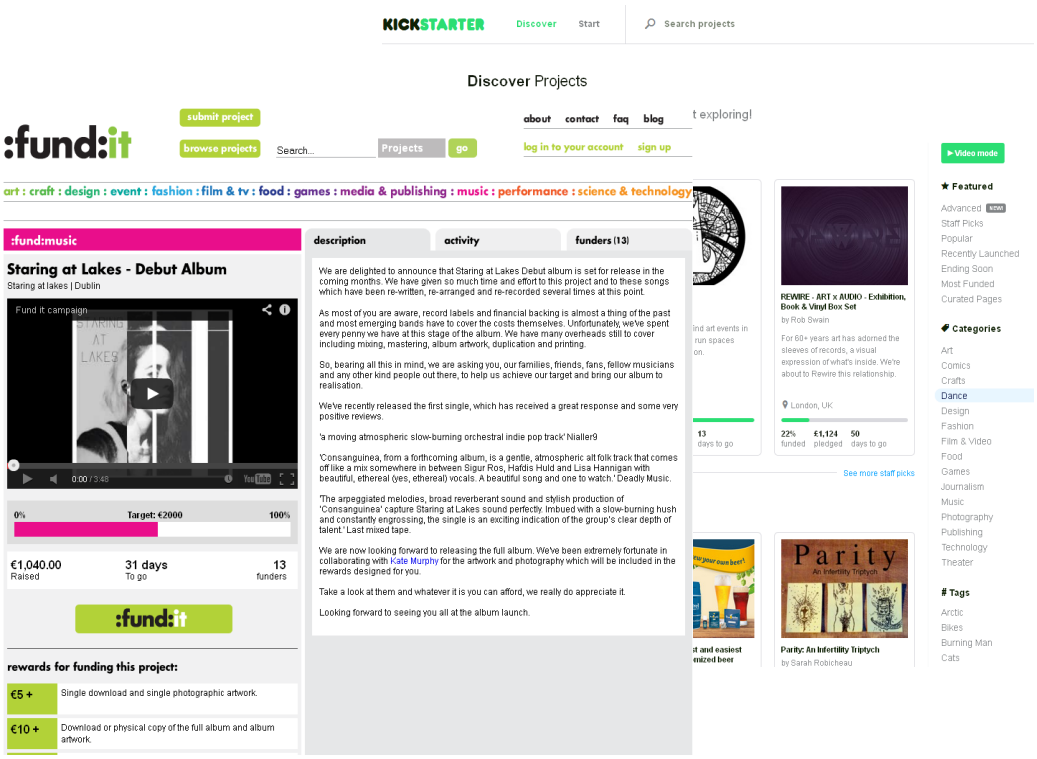
La comparativa se ha realizado entre dos webs de búsqueda de empleo, Infojobs y Bumeran, las cuáles se diferencia muy poco y son extrapolables a cualquier otra web del mismo tipo. Se han elegido las webs de empleo como paradigma web en que es posible anunciar los tipos de personas que se necesitan, en este caso para un puesto de trabajo. El inconveniente es que no tienen un entorno adecuado para intercambiar ideas y desarrollarlas ya que aunque si existe la figura del candidato y el administrador del anuncio, no hay más comunicación entre ambos que el envío de la solicitud y la posible aprobación para una entrevista.

3.1.6. Crowdfunding

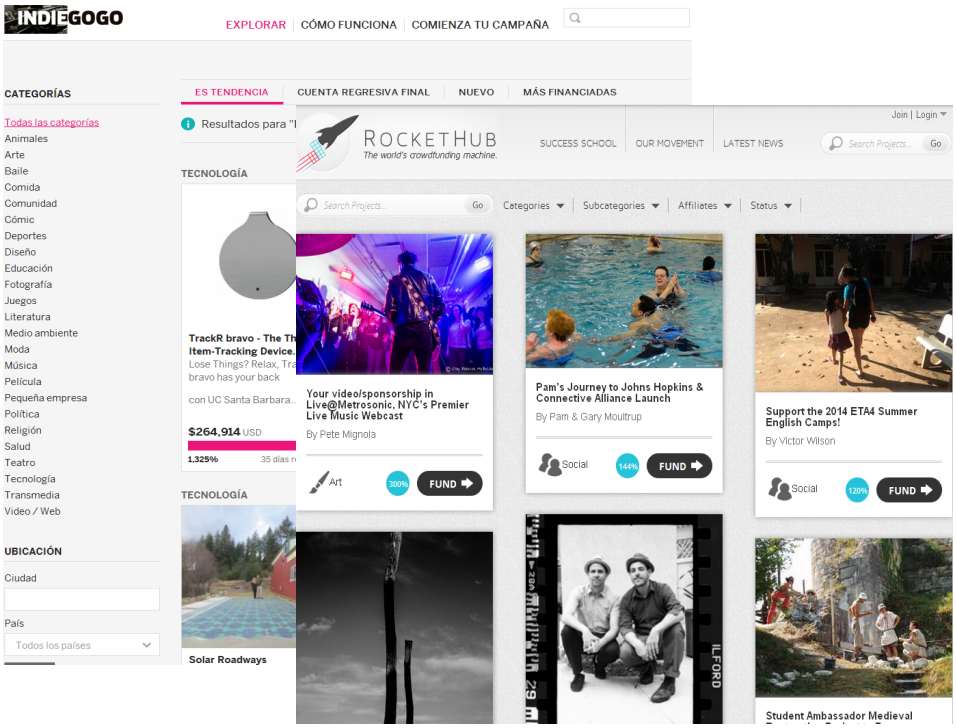
Otra característica importante de la plataforma a desarrollar es la posibilidad de financiar proyectos mediante crowdfunding o microdonaciones. El análisis ha sido realizado entre varias plataformas que han resultado ser muy similares. Coincidiendo en que mayormente ofrecen recompensas a cambio de las aportaciones e información de tanto el objetivo a recaudar como una fecha límite para ello. Casi ninguna de estas plataformas permite la asociación entre usuarios o la comunicación privada entre financiadores sino que suelen disponer de un muro de mensajes público. Como elemento diferenciador la plataforma a desarrollar transfiere los fondos directamente a la cuenta del proyecto, para así tener recursos immediatos. En cambio el resto de plataformas de crowdfunding los retienen hasta cumplir cierto criterio ya sea de tiempo o de mínimo de recursos obtenidos.

Kickstarter y Fundit disponen de varias opciones para categorizar proyectos mayormente para el ámbito de arte, cultura y tecnología, pero ninguna para el ámbito social o solidario. También incluyen un sistema de recompensas. Kiskstarter permite hacer búsquedas tanto por temática, tags o zona además de un muro de comentarios, exclusivo eso si para los financiadores. En cambio fundit solo permite búsqueda por temática, además de estar centrado exclusivamente en Irlanda.

Por tanto, kickstarter podría ser lo más parecido a la plataforma a desarrollar, pero carece de más vías de comunicación y granularidad al elegir quien puede participar en el muro de mensajes.

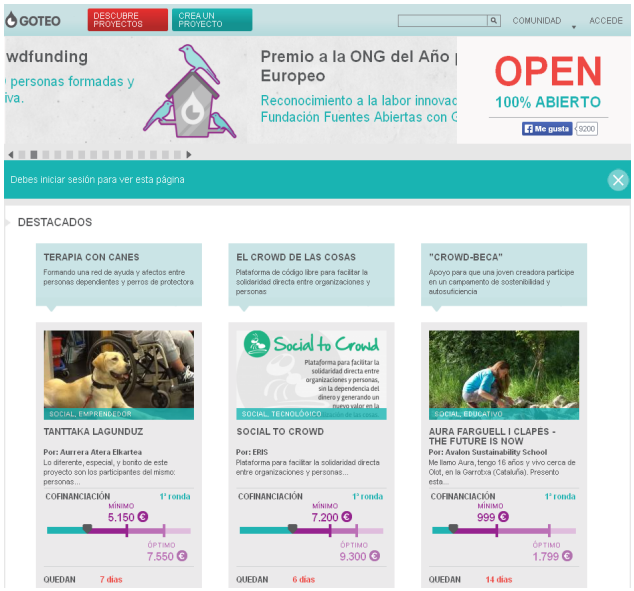


Rockethub e Indiegogo guardan gran parecido con kickstarter ofreciendo una gran cantidad de categorías. Cabe destacar que ambos incluyen categorías de ámbito social, además de emprendedor, entre otras muchas. La diferencia radica en que indiegogo si permite hacer búsquedas por zona. Sin embargo, ambos carecen de la posibilidad de elegir quien participa en la única vía de comunicación del proyecto que es también un muro.



Vorticex.org y **Teaming.net** a diferencia de los anteriores no disponen de muchas de las características, siendo portales muy sencillos. No disponen tampoco de categorización de proyectos, ni de búsquedas avanzadas como los otros. La creación del proyecto es muy rápida por su sencillez y pocos campos. Sin embargo, Vorticex.org no facilita la comunicación entre usuarios que financien el proyecto mediante muro o algún otro tipo de vía, en cambio teaming.net si, pero solo para colaboradores. En cambio vorticex si dispone de sistema de recompensas a diferencia de teaming.

Goteo.org como el resto de plataformas de crowdfunding, dispone de un sistema de financiación con recompensas, lo novedoso es que además permite anunciar necesidades no monetarias, tales como materiales, colaboradores, etc. Aunque como el resto no permite la comunicación directa y eficiente dentro de cada proyecto entre todos los colaboradores del mismo. Además, como ocurre en las otras plataformas del mismo tipo, la utilidad principal es obtener financiación pero para proyectos de los cuales ya hay conocimiento de antemano de la financiación concreta que necesita.



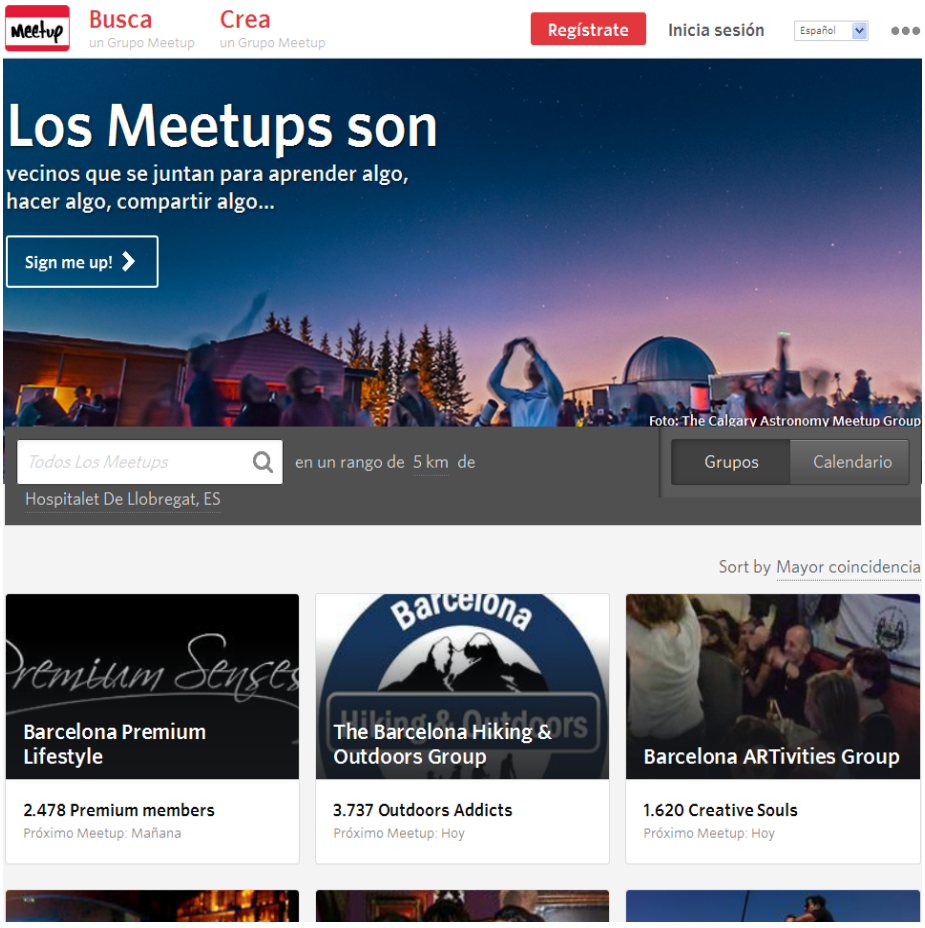
worldcoo.com está centrada en proyecto sociales y ha conseguido tener cierto renombre debido entre otras características su detallada información sobre cada proyecto, no solo descripción, sino también localización y presupuesto desglosado. Proveyendo todos estos detalles proporciona mucha confianza al usuario, además todo financiador recibe periódicamente información sobre en que es invertido el dinero recaudado.

3.1.6. Redes sociales

Existen muchas redes sociales de propósito general. Han sido escogidas solo un pequeño subconjunto y teniendo en cuenta que fueran de las más aptas para el propósito del proyecto. La gran desventaja de todas ellas es que no están preparadas para permitir la financiación, además de estar centradas solo en el encuentro de personas y no de proyectos o ideas. Aunque la incorporación de comunidades ayuda a salvar este obstáculo, siguen sin poder ofrecer un listado de gente necesaria, característica que prácticamente solo se encuentra en las webs de búsqueda de empleo. Además las pocas que disponen de grupos cerrados tienen una jerarquía de usuarios y permisos muy sencilla.

Facebook y Google+ proporcionan tanto que las personas puedan asociarse unas con otras a través de amistades o círculos como por comunidades tanto abiertas como cerradas. También ofrecen gran variedad de medios de comunicación entre usuarios, tanto vía correo, muro de la comunidad o personal, eventos y chat. Pero no ofrecen búsqueda por temática o tags. Por último, si que permiten compartir ciertos ficheros y destacar contenido. Facebook tiene la ventaja de poder clasificar los grupos en ciertas categorías, incluyendo tanto de ámbito profesional como de social, además de buscar por zona entre otros aspectos.

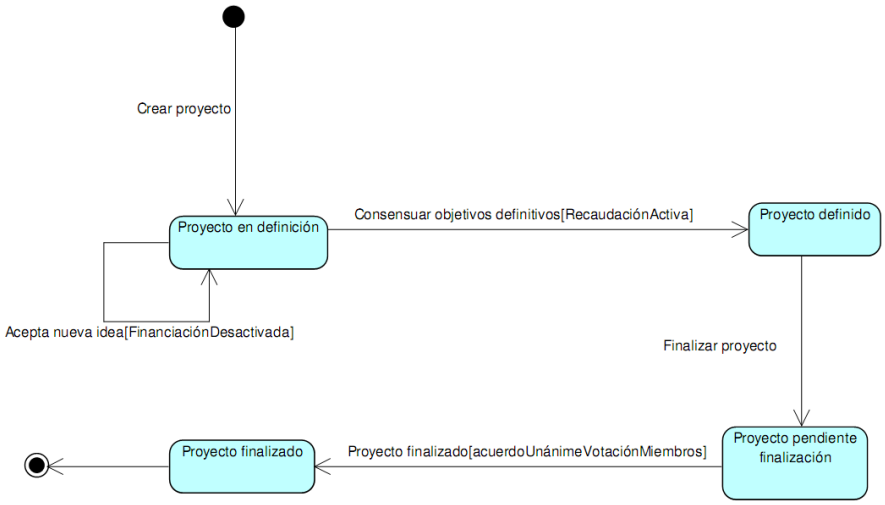
Meetup es un portal con muchas funcionalidades centradas en la creación de grupos donde conocer gente cercana para reunirse y hacer actividades de tipos muy variados. Incorpora un gran sistema de búsqueda y de clasificación por tags, temática, zona. Además los grupos pueden ser tanto de temática social, como para emprendedores entre otros. Incorpora como vías de comunicación entre usuarios, mails, eventos y a diferencia de otros, foro de discusión.



3.2. Entidades de la plataforma

La plataforma está formada por diferentes entidades. La primera y más importante es el proyecto, que contiene la información sobre la temática y objetivos. También y no menos importante, el grupo, que proporciona al proyecto las herramientas necesarias para el desarrollo de este. Otra entidad del que está compuesta la plataforma es el usuario, el cuál tendrá unos permisos diferentes según el estatus o membresía con relación a cada proyecto que tenga. Ciertos tipos de contenido, tales como el propio proyecto, pueden ser denunciados por los usuarios registrados en caso de tener contenido ofensivo. De la misma forma, es posible denunciar a usuario por comportamiento inadecuado. A continuación son presentadas las entidades de la plataforma.

3.2.1. Proyectos



Todo proyecto creado comienza como una propuesta, una idea que se convierte en un proyecto en definición y por tanto, necesita nutrirse de nuevas ideas que lo hagan crecer. La entidad proyecto contendrá la información necesaria para poder captar el interés de otros usuarios que puedan participar y ayudar a definir unos objetivos concretos, además de contribuir más adelante a la realización del proyecto. Esta información está compuesta por diversos campos tales como quiénes y con qué actividad o conocimientos específicos son necesarios, además de su descripción. Pretendemos proporcionar un lugar donde los usuarios puedan interaccionar con cierta privacidad en un entorno que facilite la comunicación por varias vías.

Una vez que el proyecto haya acabado de definirse, el administrador puede activar la financiación para empezar a captar fondos mediante crowdfunding siendo incentivado con un sistema de recompensas.

Cada proyecto está divido en dos partes bien diferenciadas y cada una de estas corresponde a una entidad, la de Proyecto y la de Grupo. La diferencia entre ambas es que el proyecto sirve para mostrar y editar la información pública del mismo, mientras que el grupo gestiona la entrada o salida de participantes, sus vías de comunicación dentro del proyecto y la financiación.

La información pública consiste en una descripción breve, útil para una rápida lectura por parte de los usuarios de la plataforma, además de una descripción completa con toda la información necesaria del mismo.

Contiene también un listado de las actividades o roles de personas necesarios para desarrollarlo. Todo proyecto está clasificado en dos tipos, emprendedor y social y además por categorías o área de conocimiento. Por último y de forma opcional, existe la posibilidad de adjuntar una imagen y/o un vídeo que complemente el resto de información introducida.

Para facilitar la criba de proyectos en los que los usuarios puedan estar interesados, los proyectos muestran la información anteriormente descrita. Teniendo en cuenta que habrá dos vistas, una en las búsquedas y proyectos recientes, donde es mostrada la descripción breve y no la completa, además de la imagen. La otra vista, modo completo, muestra la descripción completa y no la breve, además del vídeo.

Como información complementaria y siendo parte de lo que el módulo Grupo se encargue, muestra lo recaudado hasta el momento, la fecha prevista de finalización y hasta donde se ha progresado en el desarrollo del proyecto. Además, en el momento en que sea activada la financiación, es posible mostrar un desglose del presupuesto, para conocer de antemano en que será invertido lo recaudado.

Existe una actividad o rol especial, el financiador, el cual no solo permite que financie el proyecto sino que además pueda participar en él.

Los proyectos pueden ser buscados a través de un motor de búsqueda, y dispondrán de unos filtros, que se adaptarían a los resultados mostrados. Algunos de los criterios que los filtros se tienen en cuenta son el tipo de proyecto, social o emprendedor, la categoría, estado del mismo(abierto, cerrado, finalizado o eliminado), lo recaudado, el progreso conseguido y una nube de tags sobre los tags o etiquetas definidas en los propios proyectos.

Para la difusión exterior de los proyectos, éstos serán integrados con las redes sociales.

3.2.2. Usuarios

Todo usuario puede iniciar sesión con su cuenta y para ello es necesario un email y una contraseña escogida por el usuario de entre 8 y 30 caracteres con los que el usuario se registró anteriormente.

Los usuarios no registrados o anónimos pueden visualizar los proyectos y realizar búsquedas sobre estos. El resto de funcionalidades están disponibles solo para los usuarios registrados.

Los usuarios que deseen registrarse tienen que indicar un e-mail de contacto, una contraseña, un nombre usuario y una fecha de nacimiento. Siendo solo posible para usuarios a partir de 14 años (*Según legislación española) registrarse. Para asegurar que solo usuarios físicos puedan registrarse el sistema captcha es utilizado.

Todos los usuarios registrados disponen de un perfil público donde pueden indicar datos como conocimientos, formación educativa, experiencia, méritos profesionales, intereses y/o webs del mismo usuario. Además otros usuarios pueden aportar referencias sobre el usuario en cuestión teniendo que ser aprobadas por él para hacerlas públicas.

Otros datos también pueden ser públicos, si así lo indica el usuario, tales como su nombre, apellidos y dirección. La dirección puede ser empleada más adelante para poder solicitar recompensas a la hora de financiar un proyecto. En cualquier caso el nombre de usuario siempre es visible y la contraseña siempre privada.

Cada usuario registrado que cree un proyecto es el administrador único de este y por lo tanto, puede modificarlo y/o eliminarlo.

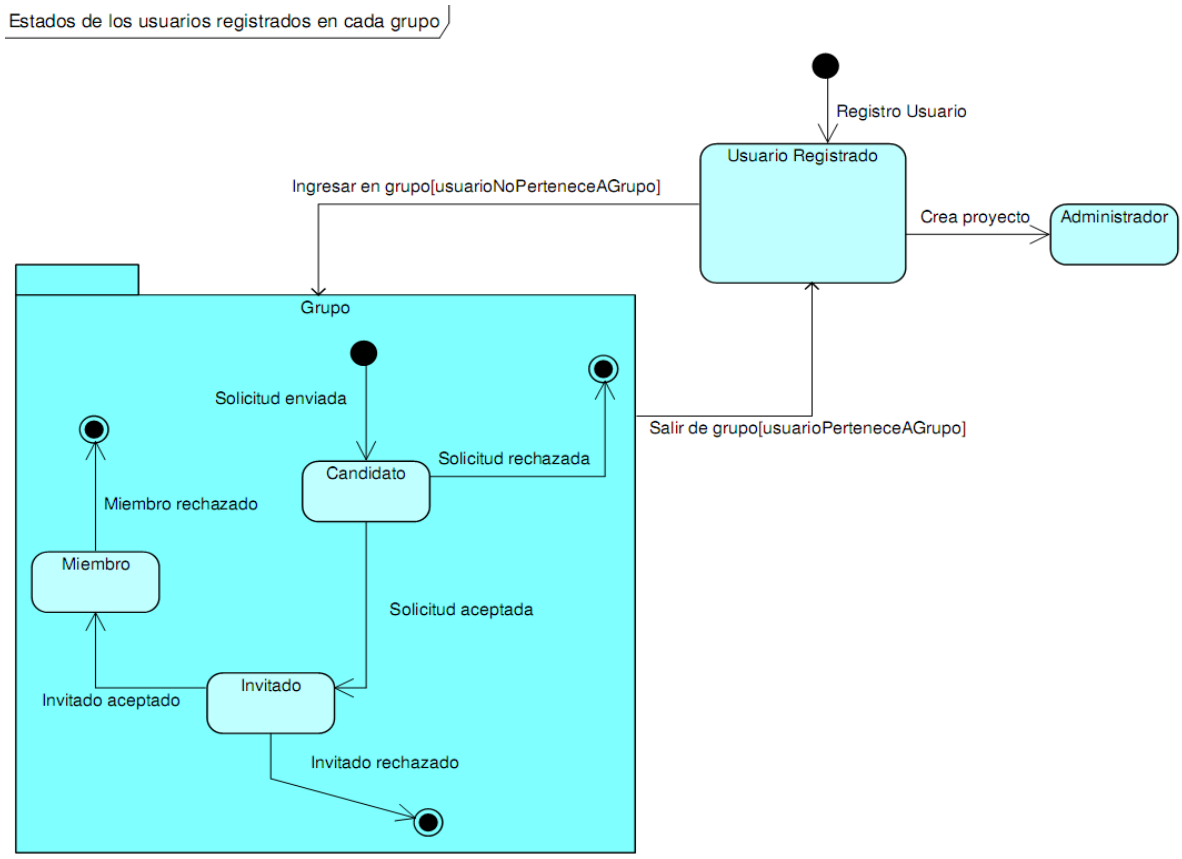
Todos los usuarios registrados pueden bloquear/desbloquear o destacar otros usuarios. Estos usuarios marcados aparecen en un listado visible solo por el usuario que los ha marcado y así poder modificar estas relaciones entre usuarios fácilmente.

Dar de baja al usuario

Cada usuario registrado podrá darse de baja del sistema, siempre y cuando haya finalizado o eliminado todos los proyectos creados por él. El resto de contenido que haya creado tales como mensajes de muro, comentarios serían asignados a un usuario anónimo de tal forma que sean conservados a pesar de que el usuario sea eliminado. En caso de haber sido expulsado de algún proyecto o estar bloqueado por algún usuario, aunque no sea posible iniciar sesión con ese usuario permanecerá en el sistema hasta que no se den las condiciones anteriormente descritas.

Tipos de usuario

Los usuarios participantes en el proyecto tendrán 4 categorías o membresías posibles dentro de éste, administrador, miembro, invitado y candidato.



El tipo de usuario principal dentro de cada proyecto es el Administrador que es el creador del proyecto. Cada usuario que requiera participar debe primero enviar una solicitud convirtiéndolo en candidato. El administrador del proyecto al aceptar esta solicitud de participación promueve al candidato hacia invitado.

Todo invitado puede comunicarse con usuarios participantes del proyecto mediante un chat exclusivo para éste. En el momento en que objetivos de invitado y participantes converjan, el administrador puede promover a éste hacia miembro, con lo cuál, tendría pleno acceso al resto de vías de comunicación del proyecto.

El administrador, si lo cree conveniente, puede expulsar a participantes (candidatos, invitados, miembros) del proyecto. Tal como se ha indicado anteriormente, permanecerán un máximo de 4 meses expulsados, a no ser que el administrador proceda a desbloquearlos. Mientras estén expulsados no pueden enviar solicitudes de entrada al proyecto.

Existe una jerarquía de permisos dependiendo de la membresía de cada usuario. Empezando por el administrador con poderes absolutos sobre este, como crear o eliminar el proyecto o datos de este, incluir o excluir miembros e invitados, además de aceptar o rechazar solicitudes de candidatos. Esto último equivalente a aceptar o rechazar candidatos. Además de acceso a todas las vías de comunicación del proyecto, configuraciones y listado de usuarios del mismo.

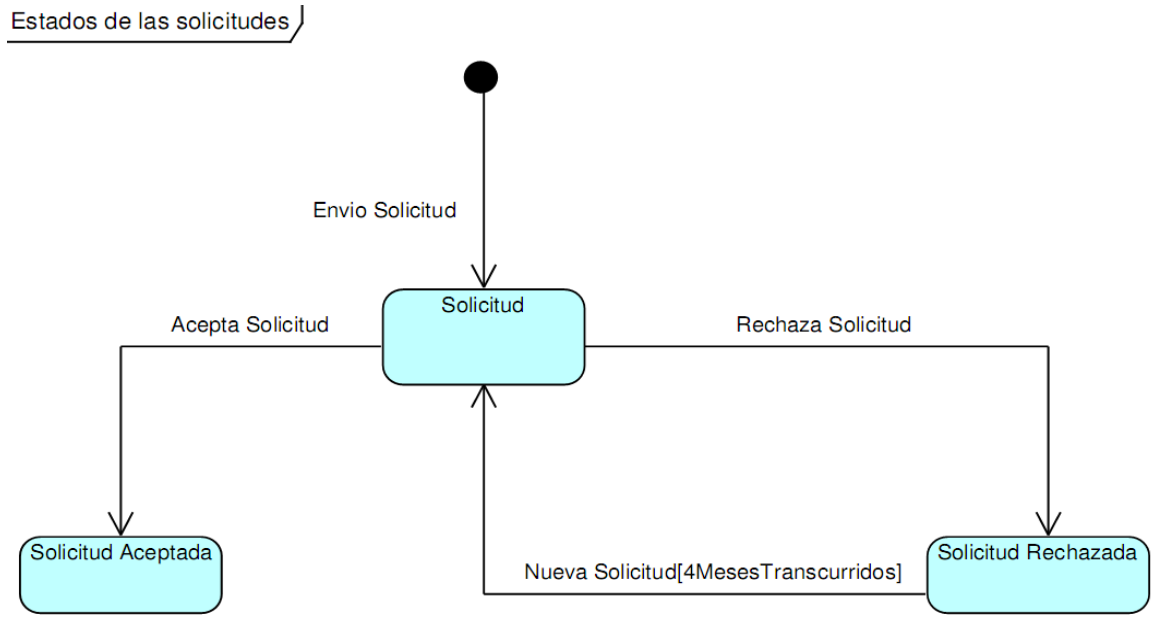
En cambio, los miembros, invitados y candidatos solo tienen la posibilidad de salir del grupo. Aunque si tienen distinto acceso a las vías de comunicación. Los invitados tienen acceso al chat. Los miembros tienen acceso a este último, además del muro y los eventos, pudiendo por tanto participar en estos.

El administrador además podrá abrir o cerrar la admisión de solicitudes para permitir o restringir que posibles candidatos puedan solicitar su ingreso en el mismo.

Todo miembro expulsado, para llegar a ser expulsado definitivamente en caso de haber realizado aportaciones económicas al proyecto tiene que recibir las devoluciones de estas. Siempre y cuando la fecha del pago no supere el periodo establecido por el gestor de transacciones bancarias escogido para realizar devoluciones. Una vez realizadas las devoluciones pasará del estado pendiente de expulsión a expulsado.

Solicitud de candidato

Los usuarios registrados que envíen una solicitud a un proyecto son convertidos en candidatos de éste. La solicitud enviada solo es visible por el administrador y el propio candidato.



Toda solicitud podrá volver a ser enviada después del periodo establecido anteriormente en caso de no haber sido contestada con una aceptación o rechazo, o eliminada. Además cada solicitud, si el proyecto lo requiere, tiene la posibilidad de pedir al candidato que aporte información extra que facilite al administrador escoger quienes formen parte del proyecto. El candidato tiene que indicar obligatoriamente el tipo de actividad o rol que piensa desempeñar, y de forma opcional que aportación económica inicial piensa realizar. Sin embargo, en caso de que el rol escogido sea de financiador tiene que indicar esa aportación inicial. Esta aportación inicial solo está disponible en caso de haber activado la financiación.

3.2.3. Vías de comunicación

Existen varias vías de comunicación entre los diferentes usuarios registrados. Estas vías de comunicación son entidades que complementan a otras tan esenciales como Grupo o Usuario. Para los proyectos existen un muro de mensajes con comentarios y un chat. Para la comunicación entre todos los usuarios de la plataforma hemos creado un sistema de correos propio

El muro de mensajes solo es accesible por administrador y miembros del proyecto. El chat es accesible tanto por administrador, miembros e invitados. A continuación son mostradas estas vías de comunicación.

Chat

El chat posibilita la comunicación instantánea entre administrador, miembros e invitados del proyecto, teniendo acceso también al listado de usuarios conectados a éste. En caso de que administrador y miembros precisen conversar sin que los invitados puedan seguir la conversación, pueden poner el chat en modo privado.

Muro

Cada mensaje del muro puede ser comentado por administrador y miembros del mismo proyecto. Además, cada autor de mensajes o comentarios puede borrarlos o editarlos y todo autor de mensaje puede eliminar los comentarios del mismo que crea conveniente. El administrador del proyecto puede eliminar mensajes o comentarios que considere ofensivos sin importar quien sea el autor de estos.

Los mensajes de muro están ordenados de más reciente a más antiguo. Los comentarios, en cambio, son mostrados por orden cronológico de más antiguo a más reciente.

Correos

Todos los usuarios registrados tienen acceso a una bandeja de correo propio de la plataforma para la comunicación directa entre ellos mismos.

Cada correo puede ser enviado a uno o más usuarios registrados, que lo recibirán en su propia bandeja de entrada. También tienen acceso a los mensajes enviados por ellos mismos. Todo correo será conservado por un máximo de 5 años, siempre y cuando el propio usuario que lo reciba o envíe no lo borre antes. Ésto no afecta al correo homólogo enviado o recibido por los otros usuarios.

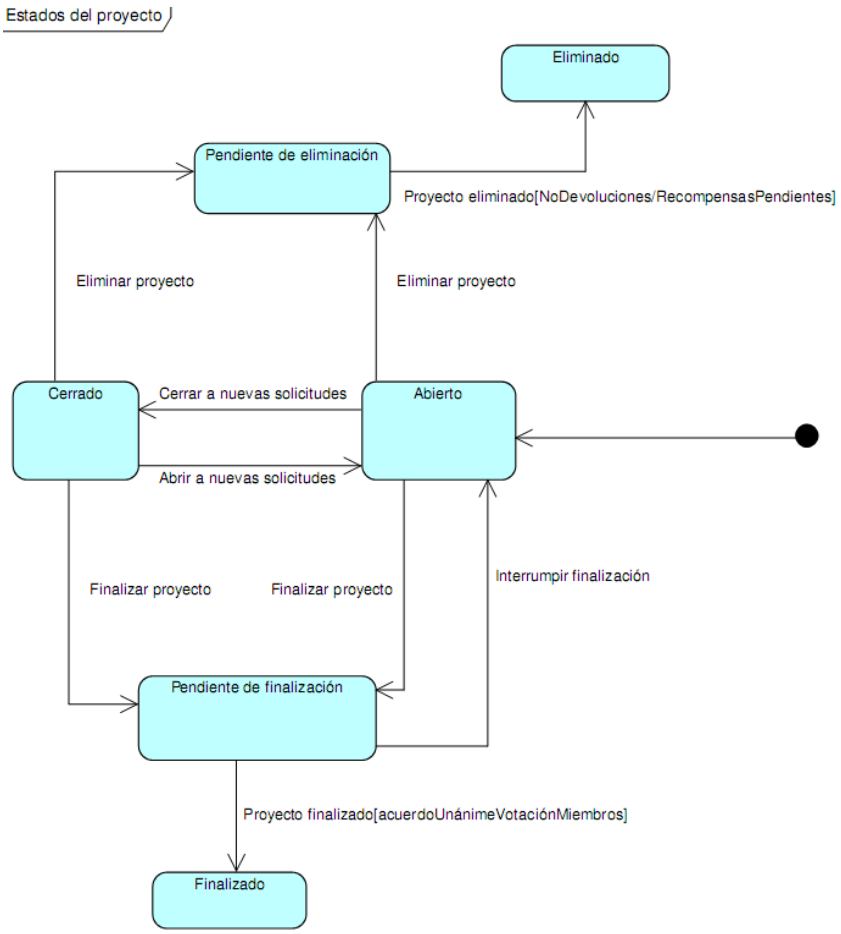
Los correos a enviar se componen de un listado de destinatarios que pueden haber sido introducidos manualmente o elegidos de un listado. Además están compuestos por un asunto y un cuerpo de mensaje. Tendremos en cuenta el listado de personas bloqueadas, ya que si existe algún destinatario del correo que tenga bloqueado al usuario emisor del correo provocaría que este destinatario no recibiese el correo.

El listado de destinatarios para elegir está compuesto por distintos apartados. Cada apartado contiene un listado de proyectos en los que el usuario emisor es administrador, miembro, invitado o candidato. También existe un apartado para los proyectos y usuarios marcados como destacados por el propio emisor.

Al elegir un proyecto en este listado son mostradas las distintas membresías visibles por el emisor. Para proyectos donde sea administrador o miembro tendrá acceso a todos los usuarios de los mismos, tanto miembros, invitados, como candidatos. En cambio en los que es invitado solo tendrá acceso a ver el administrador o miembros. Y por último para los proyectos donde es candidato solo puede ver al administrador del estos.

3.2.4. Grupos

El grupo comprende la parte privada y complementaria del proyecto. Este concentra la comunicación entre participantes, control de entrada y salida del proyecto y la financiación.



Todo proyecto podrá, siempre que el administrador lo crea conveniente, permitir recibir o dejar de recibir más solicitudes. En caso de cerrarse a nuevas solicitudes, las ya recibidas se mantendrán y podrán ser aceptadas o rechazadas de la misma forma.

Los proyectos, si el administrador así lo determina, pueden ser marcados como eliminados o finalizados, lo cuál impediría recibir nuevas solicitudes de entrada y financiación.

Todo proyecto marcado como finalizado tiene que superar una votación por parte de los miembros del proyecto, los cuáles tendrían que llegar a un acuerdo unánime. Mientras no se llegue a un consenso, el proyecto quedará en estado pendiente de finalizar, por tanto, la recepción de nuevas solicitudes y financiación quedan desactivados automáticamente. En estado finalizado las vías de comunicación como chat y muro de mensajes quedarían desactivadas, pero si que tendría una sección donde todos los usuarios puedan ver como avanza y en que son utilizados los fondos recaudados.

Todo proyecto marcado como eliminado procedería a realizar devoluciones de las aportaciones económicas recibidas, tanto de usuarios registrados como anónimos, además del envío de recompensas pendientes antes de poder ser eliminado completamente. Siempre teniendo en cuenta que las devoluciones se harían sobre pagos hechos como máximo en el periodo que establece la empresa gestora de transacciones escogida. Por tanto, mientras estén no se hayan resuelto las situaciones anteriormente descritas el proyecto quedará en estado pendiente de eliminación, desactivando así la recepción de nuevas solicitudes y financiación.

Existe un máximo permitido de candidatos, miembros e invitados por proyecto. Esto es configurable por el administrador del mismo. Los máximos iniciales para candidatos son 50, para miembros 20 e invitados 30. Aunque existen unos límites globales en la plataforma que esos límites no pueden traspasar, solo en casos excepcionales.

Estos límites globales son para candidatos 100, para miembros 40 y para invitados 60. Para ampliarlos más allá del límite tendría que solicitarlo al administrador de la plataforma.

Además es posible indicar una fecha de finalización prevista, la cual es orientativa. Esta es visible tanto por todos los usuarios de la plataforma, tanto registrados como anónimos.

El progreso del proyecto es mostrado públicamente también. Para facilitar determinar este progreso, el administrador puede crear un listado de objetivos indicando la descripción de estos junto con un porcentaje que determine en que punto del progreso del proyecto se encuentra.

Este listado de objetivos puede ser configurado para que sea visible para ciertos usuarios.

Eventos

El administrador del proyecto puede crear eventos. Cada evento está formado un título, fecha, hora, lugar y descripción del mismo. Todos los miembros del proyecto tienen acceso a los eventos de este. El administrador tiene la posibilidad de eliminar o modificar los eventos creados y cada miembro y administrador puede incluirse o excluirse de la participación en un evento.

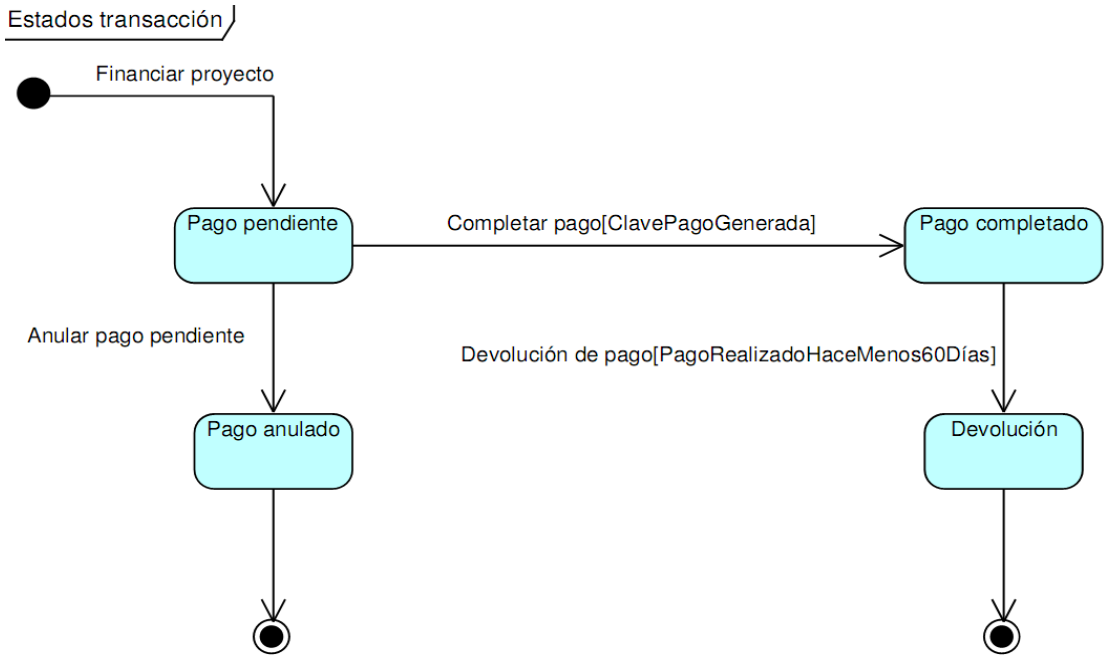
Financiación

Los proyectos que decidan obtener financiación pueden activarla, siendo necesario indicar la divisa a utilizar y la cuenta de recepción. Opcionalmente, el administrador del proyecto puede indicar el objetivo de recaudación, que es una cifra orientativa y por tanto, no impide sobrepasarla. Opcionalmente, es posible ajustar una aportación mínima para todo usuario que solicite participar en el proyecto e indique que va a financiarlo.

Además la recaudación actual de cada proyecto es visible para todos los usuarios.

Todo usuario registrado o anónimo puede realizar aportaciones económicas quedando registradas en el sistema tanto para los usuarios registrados como los anónimos. Estas quedaran en estado pendiente hasta que se completen por parte del usuario a través del gestor de transacciones.

Una vez activada la financiación, se considera que los usuarios participantes del proyecto, miembros y administrador han llegado a un consenso en la definición del proyecto y por tanto, no será posible desactivar la recompensa, a no ser en caso muy excepcionales pidiendo permiso al administrador del sistema.

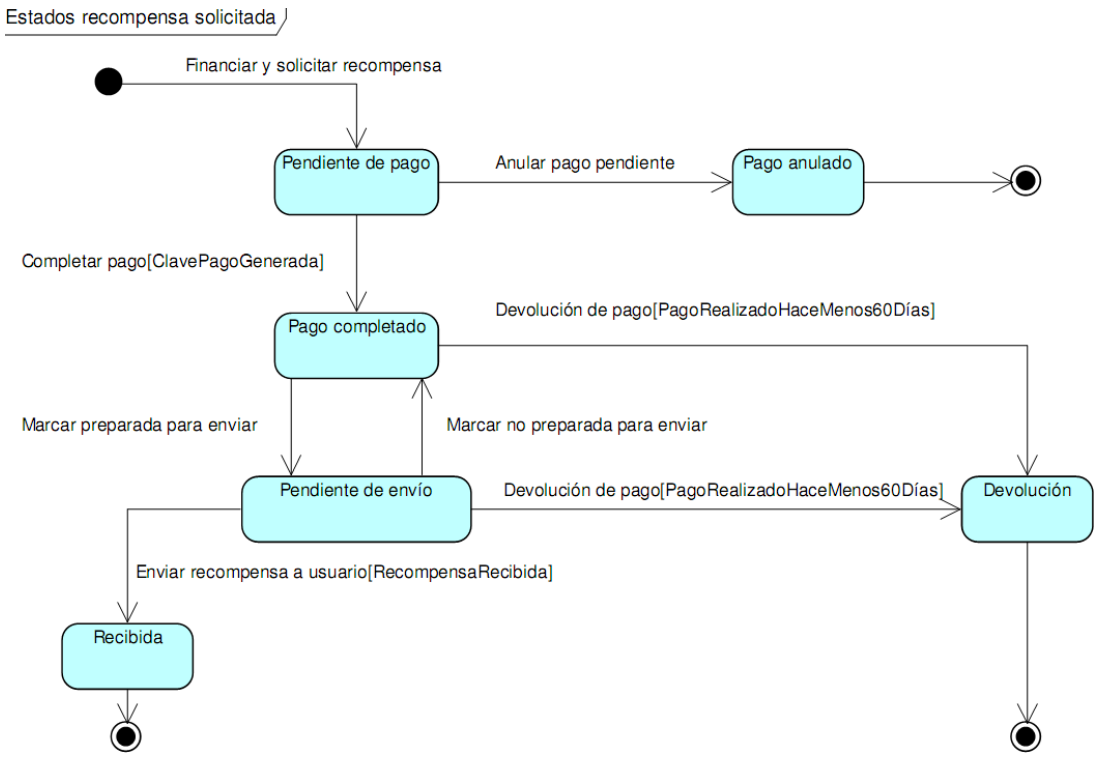


Si el usuario es administrador o miembro del proyecto, puede completar el pago en cualquier momento. En cambio si fuera candidato sólo podría anularlo o completarlo una vez llegue a miembro. Por tanto, el invitado tendría que esperar a llegar a ser miembro para completar el pago ya que no puede anularlo. Esta diferencia es debida a que un candidato promovido a invitado y que en su solicitud había indicado una aportación económica concreta, está prometiendo que contribuirá con esa cantidad al aceptar su solicitud y por tanto, una vez sea invitado ya no puede anularla. El hecho de que tengan que esperar a ser miembros para completar el pago es debido a que, tal como se indicó anteriormente, reduce en gran medida la posibilidad de que sean realizadas devoluciones con demasiada frecuencia. Ya que todo usuario que llega a miembro es mucho menos probable que sea expulsado que un invitado o candidato ya que ha conseguido pasar más filtros.

También es importante añadir que los miembros aceptados, que anteriormente como invitado o candidato prometieron una financiación inicial estarían en el estado pendiente de aceptación hasta completar el pago. En este estado pendiente de aceptación tendrían los mismos permisos que cualquier otro invitado, pero una vez completado el pago participarían realmente como miembros.

Cada proyecto puede tener un listado de recompensas. Las recompensas tienen configuradas una cifra mínima necesaria para solicitarla, una descripción breve, el número estimado de semanas que tardaría en estar preparada y el número límite de recompensas que a la vez pueden estar solicitadas.

Incluyendo con esto último las que están pendientes de pago, las que tienen el pago completado pero aun no están preparadas para enviar, las que están preparadas para enviar y las pendientes de devolución del pago.



Las recompensas ya solicitadas son visibles tanto por el administrador del proyecto como por el usuario solicitante. De esta forma, el administrador podrá determinar cuando una recompensa tiene el pago completado y puede estar lista para ser enviada al usuario. Una vez que el usuario la reciba, este puede marcarla como recibida y así ser eliminada del sistema.

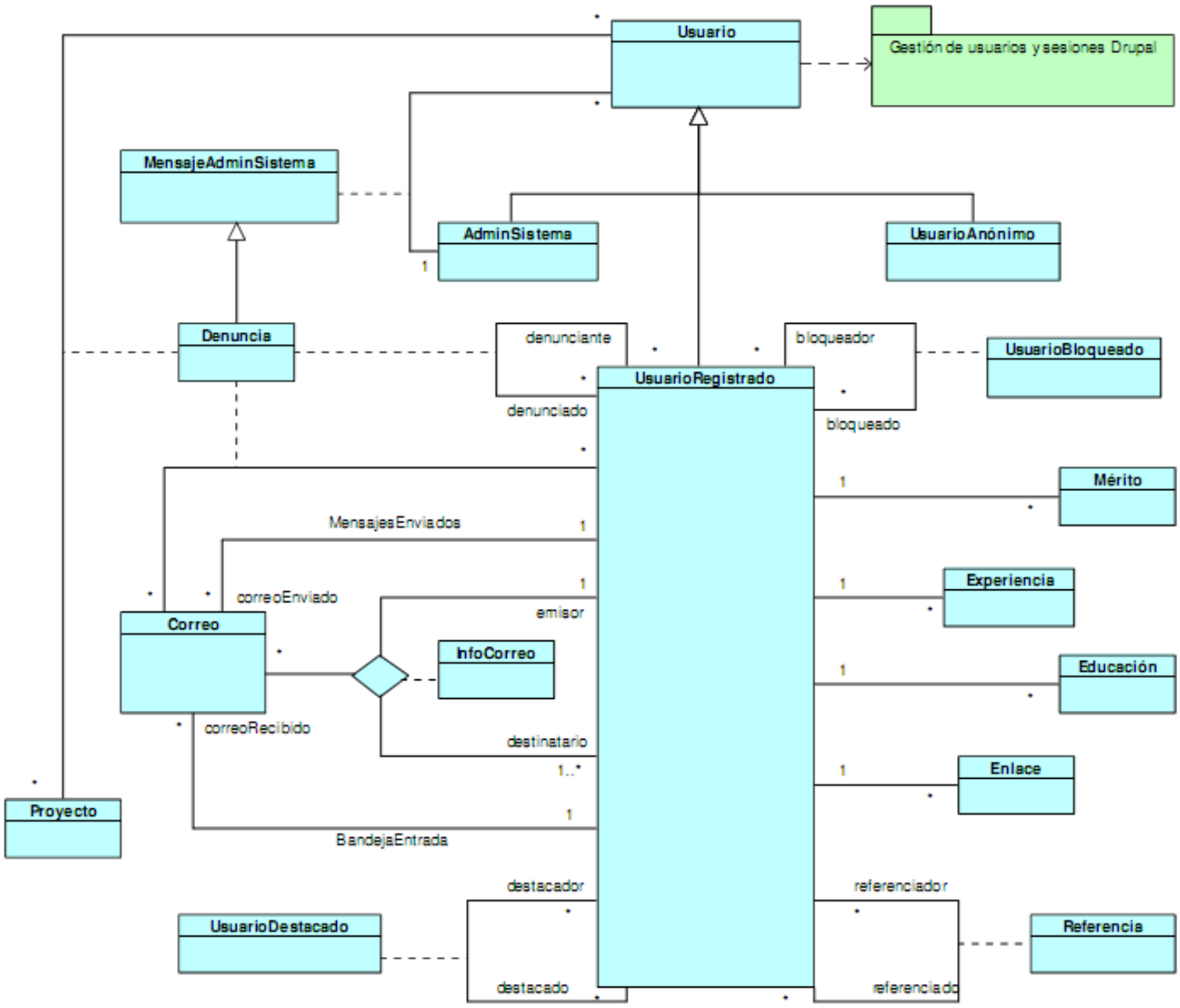
Toda recompensa solicitada puede ser eliminada y por tanto, el usuario no la recibirá, siempre y cuando, esta tenga el pago pendiente, de forma que no solo será eliminada la recompensa sino también el pago.

También podrá ser eliminada aunque tenga el pago completado siempre y cuando entre dentro del periodo establecido por la empresa gestora de transacciones bancarias escogida para realizar devoluciones. En este último caso, la recompensa pasará al estado pendiente de devolución de pago antes mencionado.

3.3. Diagrama de clases

A continuación es mostrado el diagrama de clases, que para mayor claridad está dividido en varías partes. El primer diagrama centrándose en el usuario. Seguidamente son presentados los diagramas que focalizan proyecto y grupo. Han sido divididos también en varios diagramas, uno centrándose en aspectos generales y otro en la asociación de las distintas membresías con el grupo. Y por último presentamos los diagramas de las vías de comunicación, tales como muro, chat e incluso eventos. En algunos diagramas aparecen clases de color verde, las cuáles significan que son parte de otro sistema ajeno a lo que hemos realizado en el proyecto pero necesarias para el desarrollo de este. Los diagramas completos con atributos se encuentran en el Apéndice I – Diagramas de clases completos.

3.3.1. Diagrama de clases de Usuario



Restricciones textuales:

1 - En lista de bloqueados de cada usuarioRegistrado no podrá estar incluido él mismo.

```
context UsuarioRegistrado inv
    self.bloqueo[bloqueado] -> excludes(self)
```

2 - En lista de destacados cada usuarioRegistrado no podrá estar incluido él mismo.

```
context UsuarioRegistrado inv
    self.destacado[destacado] -> excludes(self)
```

3 - Una referencia de un UsuarioRegistrado no puede estar dirigida a él mismo.

```
context UsuarioRegistrado inv
    self.referencia[referenciado] -> excludes(self)
```

4 - Una denuncia de un UsuarioRegistrado no puede estar dirigida a él mismo.

```
context UsuarioRegistrado inv
    self.denuncia[denunciado] -> excludes(self)
```

5 - Los usuarios registrados solo podrán denunciar correos donde sean destinatarios.

```
context Denuncia inv
    self.correo.destinatario -> includes(self.usuarioRegistrado)
```

6 - Todo correo recibido tendrá como destinatario al usuario del correo recibido

```
context UsuarioRegistrado inv
    self.correoRecibido ->forAll(c | c.infoCorreo.destinatario -> includes(self))
```

7 - Todo correo enviado tendrá como emisor al usuario del correo enviado

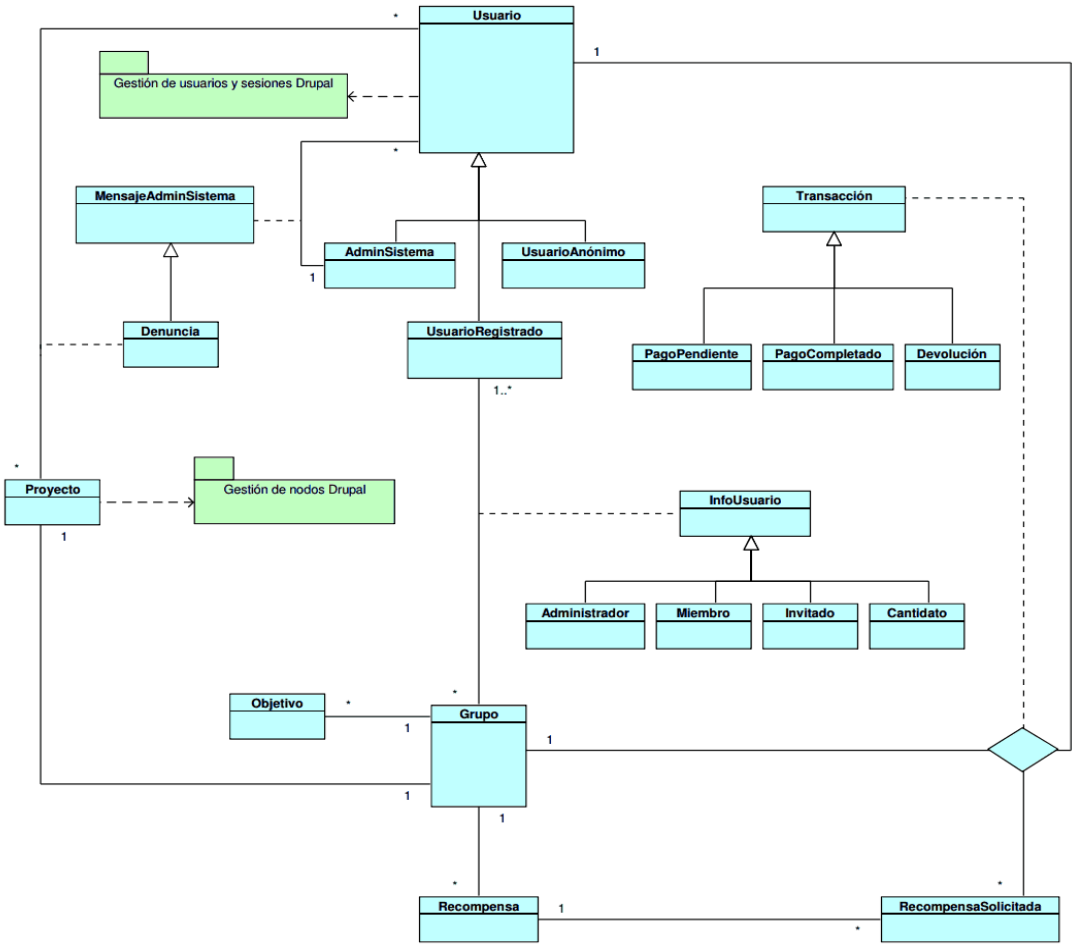
```
context UsuarioRegistrado inv
    self.correoEnviado -> forAll(c | c.infoCorreo.emisor = self)
```

La clase usuario está dividida en tres subclases, la de UsuarioAnónimo que son todos los usuarios visitantes que no inician sesión, la de UsuarioRegistrado que son los usuarios que si han iniciado sesión y por último el AdminSistema que es el superusuario que se encarga de configurar y mantener la plataforma. Éste último aunque tal como está construido Drupal es realmente un usuarioRegistrado con privilegios, por claridad ha sido separado en una clase aparte.

Todo usuario registrado puede tener información pública en el perfil tales como enlaces a sus páginas web, experiencia profesional, formación educativa, mérito o premios y por último referencias que otros usuarios le hagan. Además tiene una lista de usuarios a los que ha destacado y otra para los que ha bloqueado.

Los usuarios registrados tienen una bandeja de entrada y una de salida para los correos. Además tanto los correos, como los usuarios podrán ser denunciados.

3.3.2 Diagrama de clases General de Grupo



Restricciones textuales:

1 - El creador de un Proyecto será el administrador del grupo asociado a ese Proyecto

```
context Grupo inv
    self.proyecto.creador -> includes(self.administrador.usuarioRegistrado)
```

2 - Toda transacción que tenga una recompensa solicitada deberá ser del mismo grupo al que pertenece la recompensa

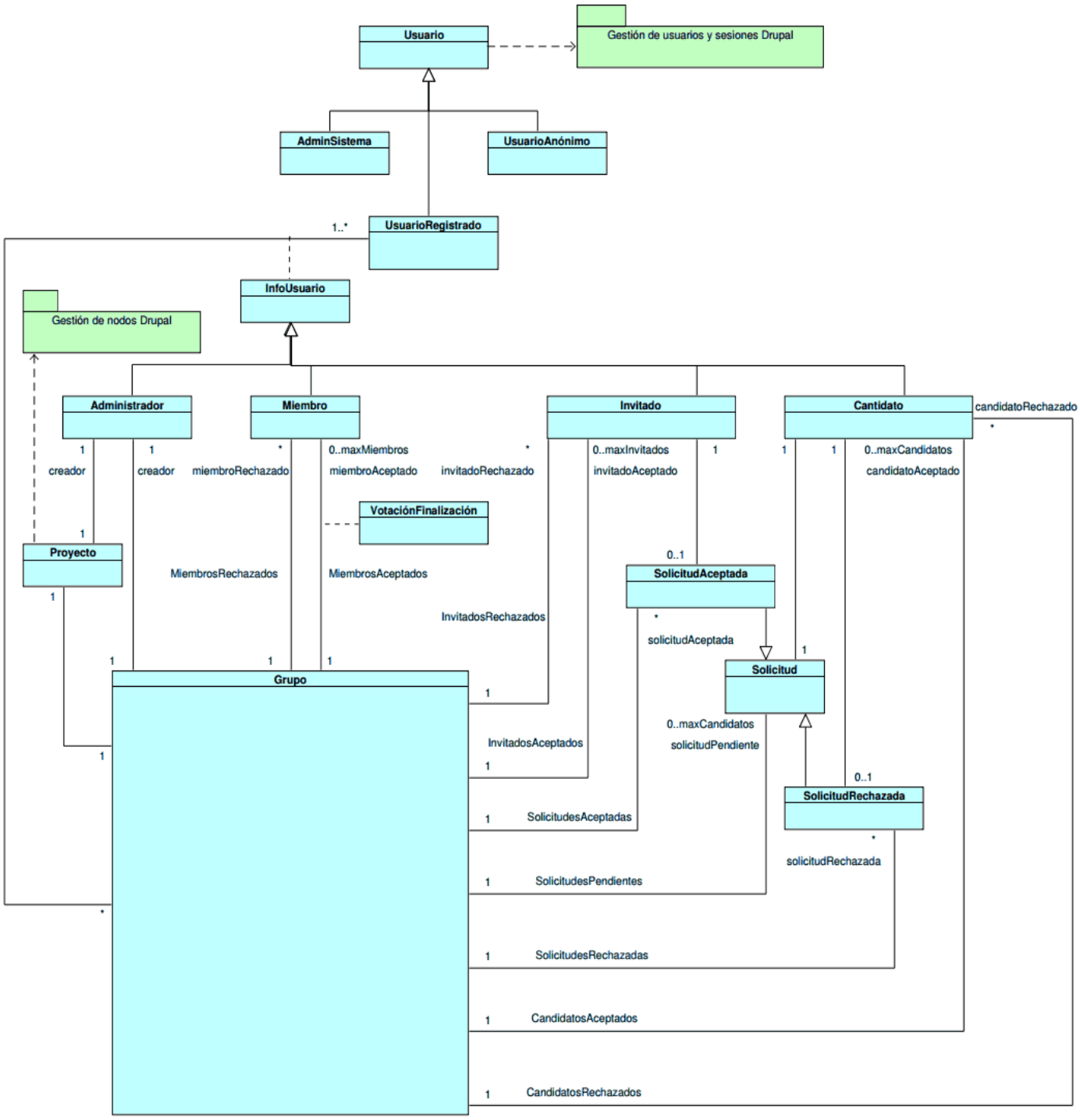
```
context Transacción inv
    self.grupo = self.recompensaSolicitada.recompensa.grupo
```

3 - Todo usuario registrado solo podrá tener como máximo un pago pendiente por grupo

```
context UsuarioRegistrado inv
    self.pagoPendiente.grupo -> forAll(g1, g2 | g1 <> g2)
```

Cada proyecto tiene un solo grupo y este grupo puede tener varios objetivos y recompensas creadas.
Toda transacción tanto sea un pago pendiente de completar, un pago completado o una devolución es de un proyecto/grupo y usuario. Además esta transacción puede tener una recompensa solicitada de las que se ofrecen el proyecto.Los proyectos pueden ser denunciados por los usuarios tanto registrados como anónimos.

3.3.3. Diagrama de clases de Asociación de Grupo



Restricciones textuales:

1 - El creador de un Proyecto será el administrador del grupo asociado a ese Proyecto

```
context Grupo inv
    self.proyecto.creador -> includes(self.administrador.usuarioRegistrado)
```

2 - Un Candidato de un grupo tendrá enviadas, a la vez, una solicitud a ese mismo grupo

```
context Grupo inv
    self.candidatoAceptado -> intersection(self.solicitudPendiente.cantidato) -> asSet() -> size() = 1
```

3 - Una SolicitudAceptada de un invitado sserá del mismo grupo al que pertenece el invitado

```
context Grupo inv
    self.invitadoAceptado.solicitudAceptada -> forAll(s | s -> intersection(self.solicitudAceptada) -> notEmpty())
```

4 - Una SolicitudRechazada de un candidato rechazado será del mismo grupo al que pertenece el candidato rechazado.

```
context Grupo inv
    self.solicitudRechazada.candidato -> forAll(c | c -> intersection(self.candidatoRechazado) -> notEmpty())
```

5 - Todo participante aceptado por un grupo no puede estar rechazado a la vez

```
context Grupo inv
    self.candidatoAceptado -> intersection(self.candidatoRechazado) -> isEmpty() and
    self.invitadoAceptado -> intersection(self.invitadoRechazado) -> isEmpty() and
    self.miembroAceptado -> intersection(self.miembroRechazado) -> isEmpty()
```

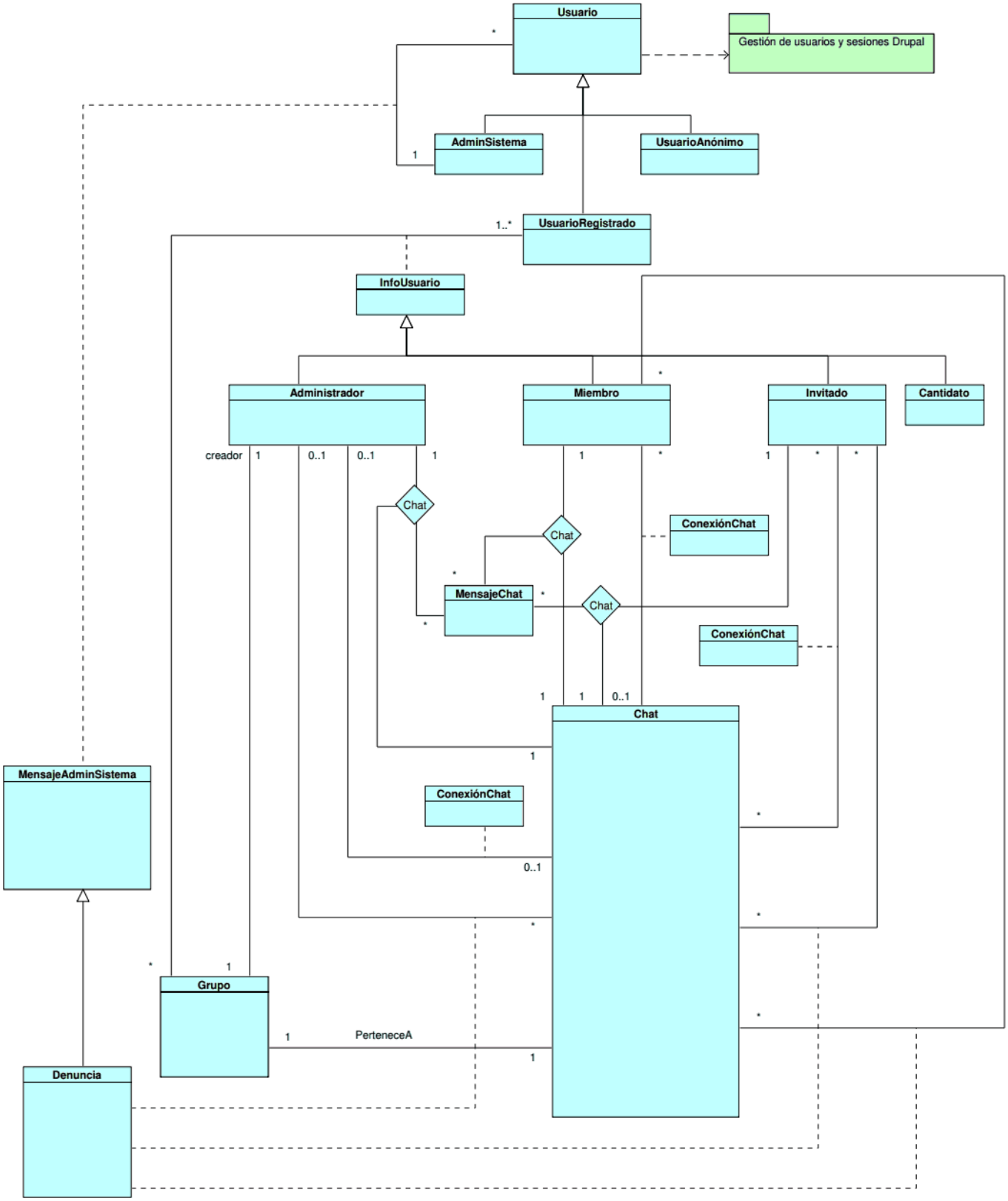
6 - Todo participante de un grupo solo puede ser de un TipoParticipante con relación a éste

```
context Grupo inv
    self.candidatoAceptado -> intersection(self.invitadoAceptado-> union(self.miembroAceptado)) -> isEmpty() and
    self.invitadoAceptado -> intersection(self.candidatoAceptado-> union(self.miembroAceptado)) -> isEmpty() and
    self.miembroAceptado -> intersection(self.candidatoAceptado-> union(self.invitadoAceptado)) -> isEmpty() and
    self.candidatoRechazado -> intersection(self.invitadoRechazado-> union(self.miembroRechazado)) -> isEmpty() and
    self.invitadoRechazado -> intersection(self.candidatoRechazado-> union(self.miembroRechazado)) -> isEmpty() and
    self.miembroRechazado -> intersection(self.candidatoRechazado-> union(self.invitadoRechazado)) -> isEmpty()
```

Cada grupo tiene un administrador o creador y puede tener varios miembros, invitados y candidatos aceptados. En caso de ser estos expulsados pasan a estar expulsados pero conservando el nombre de la membresía con la que participaban para así facilitar al administrador la búsqueda de estos en caso de querer desbloquearlos. Todo candidato tiene una solicitud enviada al proyecto en caso de ser aceptada es promovido a invitado, en caso de ser rechazada, el candidato pasa a ser rechazado.

En caso de que el administrador decida finalizar el proyecto debe someterlo a votación entre los miembros aceptados.

3.2.4. Diagrama de clases de Chat



Restricciones textuales:

1 - El administrador, miembro o invitado de un grupo solo podrá escribir en el chat del grupo al que pertenezcan

```
context Chat inv
        self.administrador = self.grupo.administrador and self.grupo.miembroAceptado -> includesAll(self.miembro) and self.grupo.invitadoAceptado -> includesAll(self.invitado)
```

2 - El administrador, miembro o invitado de un grupo solo podrá conectarse al chat del grupo al que pertenezcan

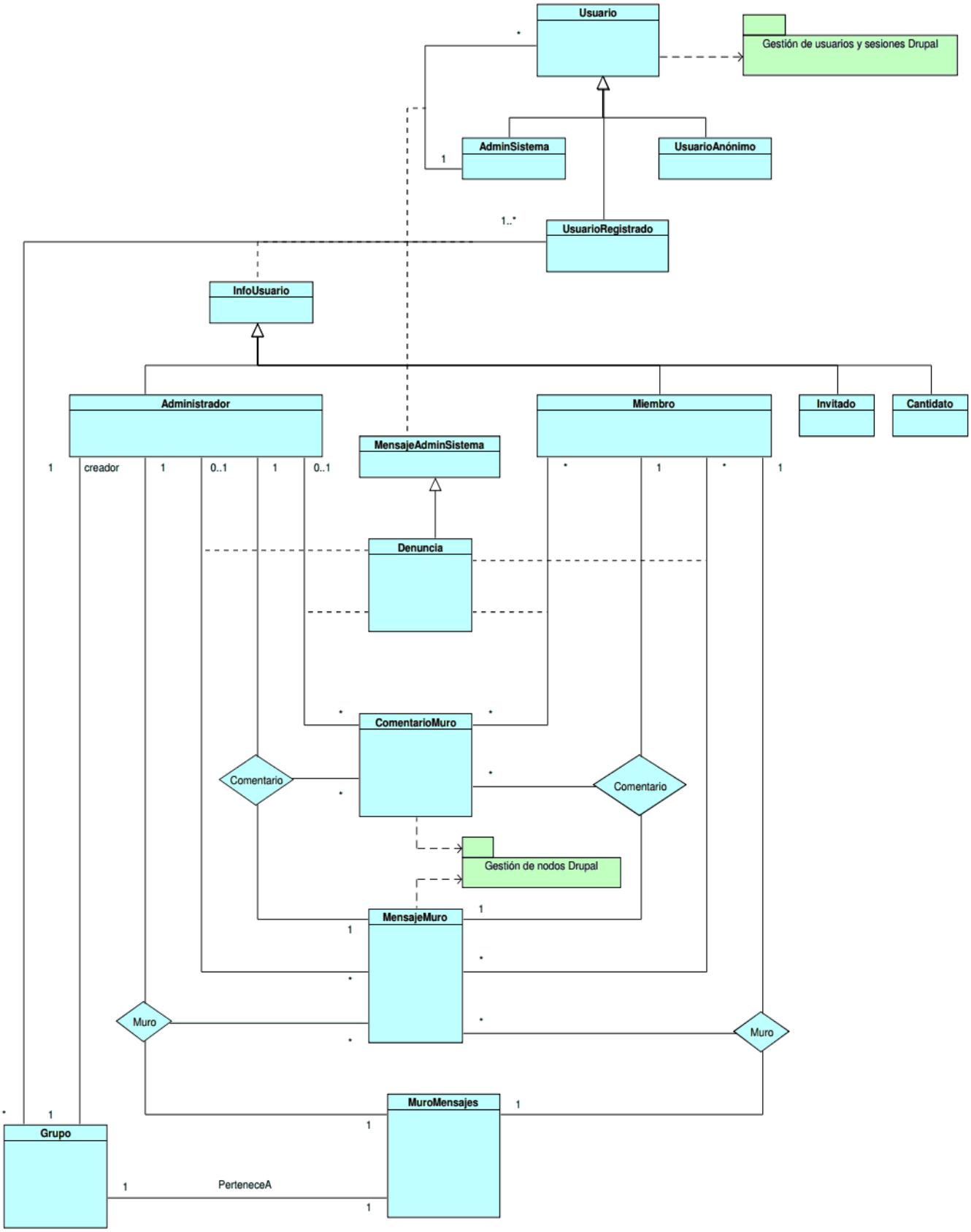
```
context ConexiónChat inv
    self.administrador = self.chat.grupo.administrador and self.chat.grupo.miembroAceptado -> includes(self.miembro) and self.chat.grupo.invitadoAceptado -> includes(self.invitado)
```

3 - El administrador, miembro o invitado de un grupo solo podrá denunciar el contenido de un chat del grupo al que pertenezcan

```
context Denuncia inv
  self.usuarioRegistrado.administrador.grupo -> includes(self.chat.grupo) or
    self.usuarioRegistrado.miembro.grupo -> includes(self.chat.grupo) or
    self.usuarioRegistrado.invitado.grupo -> includes(self.chat.grupo)
```

Cada grupo tiene un chat solo accesible para administrador, miembros e invitados. Estos pueden además denunciarlo en caso de que haya contenido ofensivo. Además indicar que cada vez que inician el chat se establece una conexión, de esta forma el resto de usuarios, con permisos para entrar en el chat, pueden saber quien está conectado en ese momento.

3.2.5. Diagrama clases de Muro



Restricciones textuales:

1 - El administrador o miembro de un grupo solo podrá escribir en el muro de mensajes del grupo o grupos a los que pertenezcan

```
context MuroMensajes inv
  self.administrador = self.grupo.administrador and self.grupo.miembroAceptado -> includesAll(self.miembro)
```

2 - El administrador o miembro de un grupo solo podrá comentar mensajes del muro del grupo al que pertenezcan

```
context Comentario inv
  self.administrador = self.grupo.administrador and self.grupo.miembroAceptado -> includesAll(self.miembro)
```

3 - El administrador o miembro de un grupo solo podrá denunciar mensajes de muro del grupo o grupos a los que pertenezcan

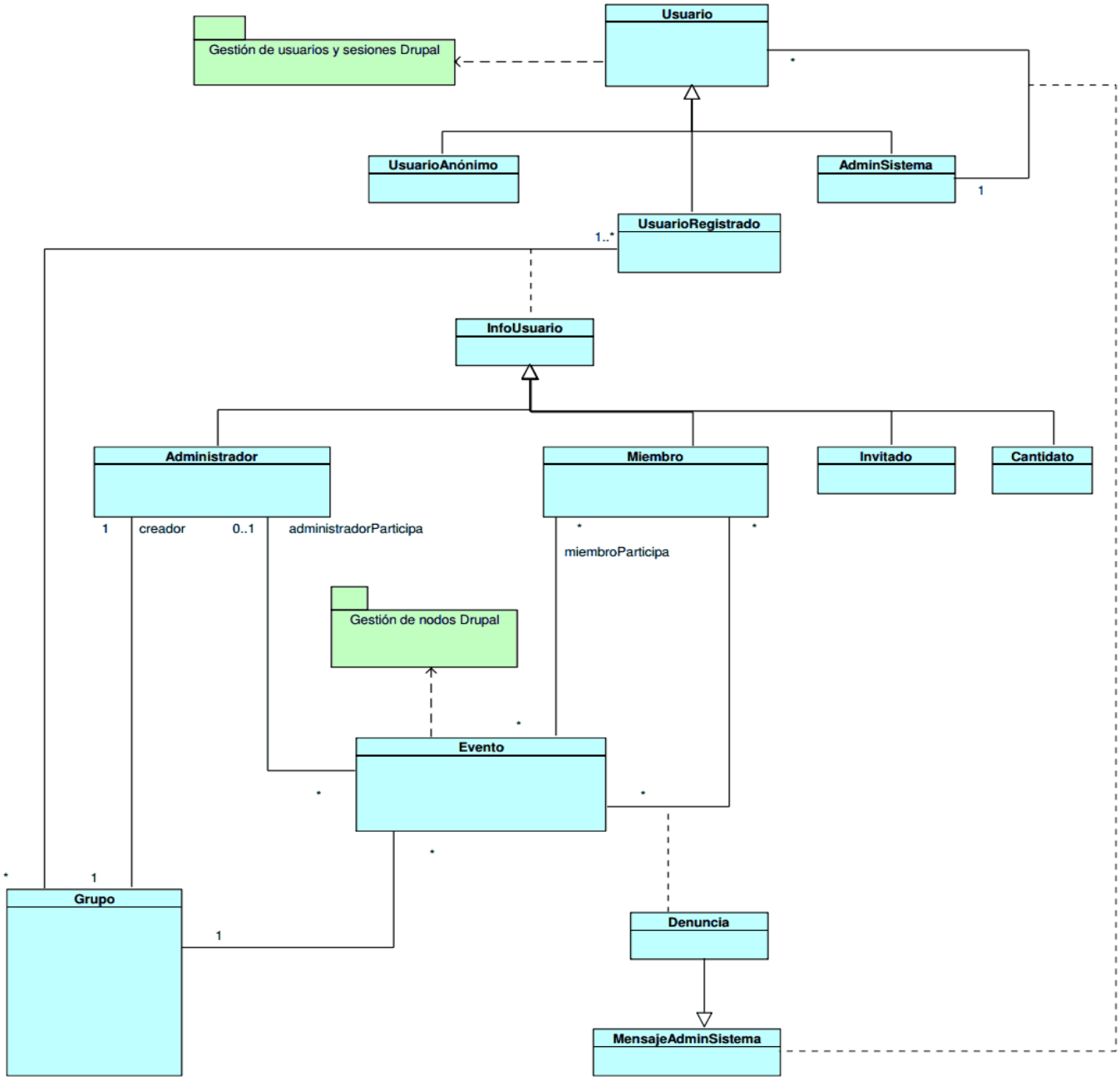
```
context Denuncia inv
  self.usuarioRegistrado.administrador.grupo -> includes(self.mensajeMuro.muroMensajes.grupo) or
    self.usuarioRegistrado.miembro.grupo -> includes(self.mensajeMuro.muroMensajes.grupo)
```

4 - El administrador o miembro de un grupo solo podrá denunciar comentarios de mensajes del muro del grupo o grupos a los que pertenezcan

```
context Denuncia inv
  self.usuarioRegistrado.administrador.grupo ->
      includes(self.comentarioMuro.mensajeMuro.muroMensajes.grupo) or
    self.usuarioRegistrado.miembro.grupo ->
      includes(self.comentarioMuro.mensajeMuro.muroMensajes.grupo)
```

Cada grupo tiene un muro de mensajes solo accesible para administrador y miembros. Este muro de mensajes está compuesto por los mensajes de muro enviados estos usuarios. A su vez cada mensaje de muro puede tener un listado de comentarios asociado escritos también por estos usuarios. Estos pueden además denunciar tanto mensajes como comentarios en caso de que haya contenido ofensivo.

3.2.6. Diagrama de clases de Eventos



Restricciones textuales:

1 - En cada evento solo podrán participar administrador y miembros del mismo grupo al que pertenece el evento.

context Evento inv
self.administradorParticipa.grupo = self.grupo and self.miembroParticipa.grupo -> asSet() =
evento.grupo

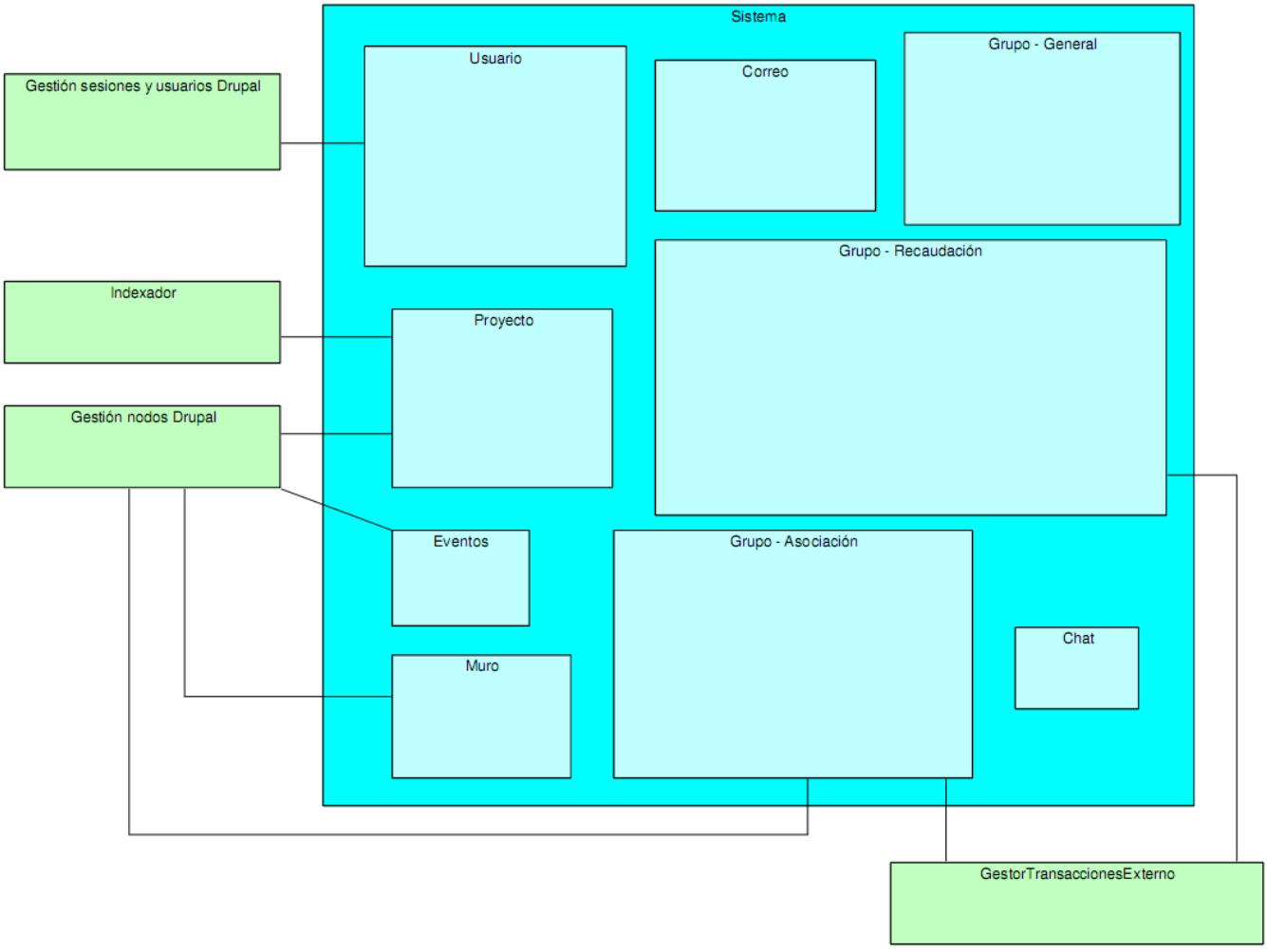
2 - El miembro de un grupo solo podrá denunciar los eventos del grupo al que pertenezca

context Denuncia inv
self.usuarioRegistrado.miembro.grupo -> includes(self.evento.grupo)

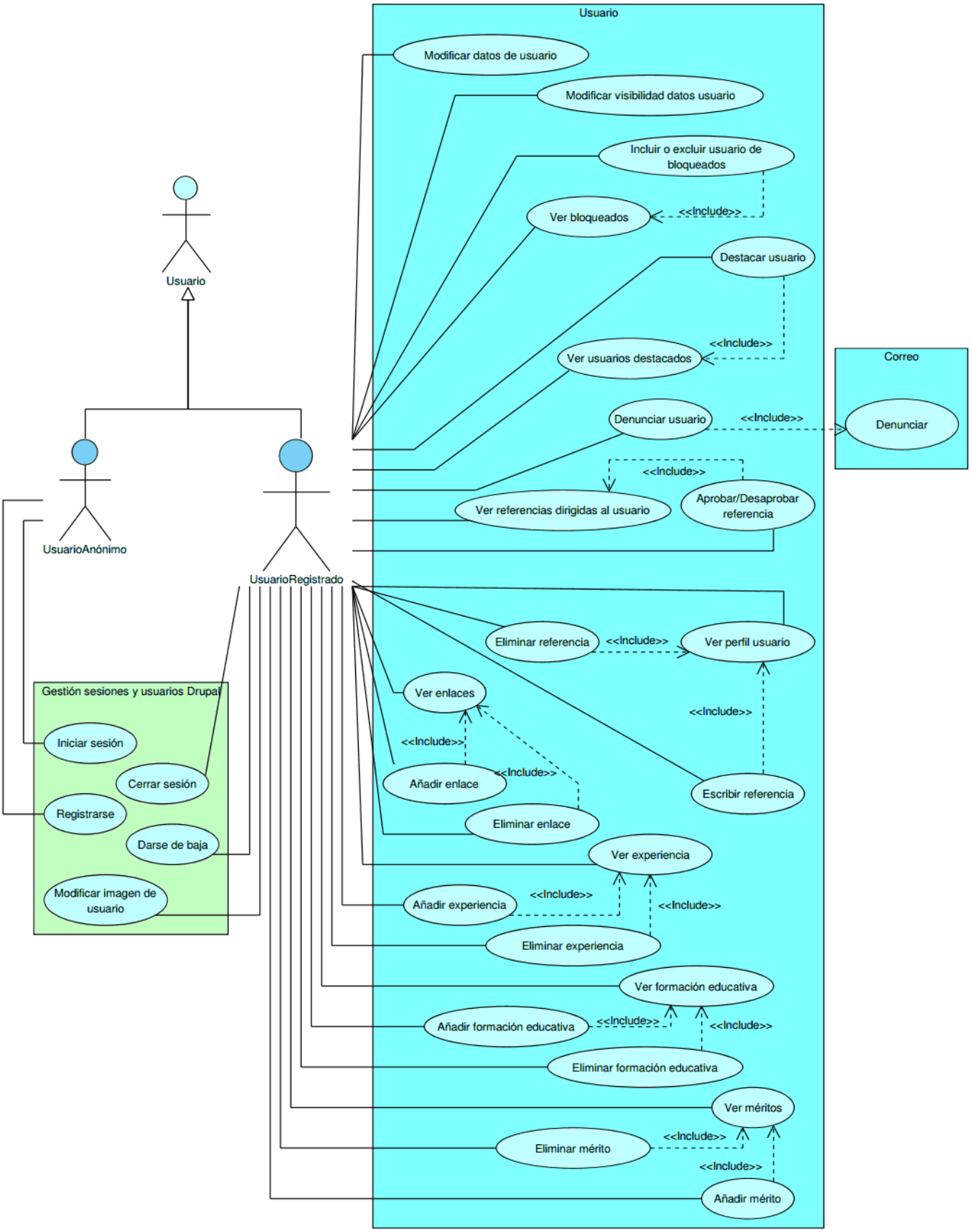
3.3. Casos de uso

Hemos agrupado los casos de uso en varios subsistemas, presentados en el diagrama de debajo para facilitar la lectura de los casos de uso. Los subsistemas en color verde pertenecen a sistemas externos, ya sea drupal, apachesolr como indexador o Paypal como gestor de transacciones.

Los actores Administrador, Miembro, Invitado y Candidato aparecen en la jerarquía de actores por debajo de usuario registrado. Aunque en la práctica no es exactamente así, con esto se quiere representar que el usuario registrado actuando en un proyecto en particular como una de esas membresías anteriormente mencionadas tiene acceso a ciertos casos de uso tal como está representado en los diagramas.



3.3.1. Casos de uso de Usuario



Modificar datos de usuario

Actores. UsuarioRegistrado
Descripción. Modifica los datos de usuario personales tales como nombre, apellidos y dirección.
Precondiciones. En caso de indicar una contraseña, esta ha de tener una longitud entre 8 y 30 caracteres.
Postcondiciones. Modifica los datos personales que el usuario haya solicitado modificar.
Escenario principal.

- El usuario indica los datos personales a modificar.
- El sistema registra las modificaciones en los datos indicados por el usuario.

Modificar visibilidad de datos personales

Actores. UsuarioRegistrado
Descripción. Modifica la visibilidad de los datos del usuario hacia el resto de usuarios.
Precondiciones. -
Postcondiciones. La visibilidad de los datos que el usuario ha indicado queda registrada en el sistema.
Escenario principal.

- El usuario indica los datos a los que modificará la visibilidad tales, como email, fecha de nacimiento, datos personales o proyectos en los que participa.
- El sistema registra las modificaciones de la visibilidad de los datos indicados por el usuario.

Ver bloqueados

Actores. UsuarioRegistrado
Descripción. Muestra los usuarios que el propio usuario ha bloqueado.
Precondiciones. -
Postcondiciones. Muestra el listado de usuarios bloqueados por el usuario.
Escenario principal.

- El usuario solicita el listado de usuarios bloqueados.
- El sistema genera el listado de usuarios bloqueados por el usuario.

Incluir o excluir usuario de bloqueados

Actores. UsuarioRegistrado
Descripción. Incluye o excluye un usuario de su propia lista de usuarios bloqueados.
Precondiciones. El usuario a bloquear o desbloquear no puede ser él mismo.
Postcondiciones. Incluye al usuario indicado en la lista de bloqueados si no lo estaba o lo excluye si ya lo estaba. En caso de incluirlo si estaba destacado deja de estarlo.
Escenario principal.

- Ver bloqueados.
- El usuario solicita incluir o excluir al usuario indicado como bloqueado.
- El sistema incluye al usuario en la lista de bloqueados propios si no lo estaba o lo excluye si ya lo estaba. En caso de incluirlo si estaba destacado deja de estarlo.

Ver usuarios destacados

Actores. UsuarioRegistrado
Descripción. Muestra los usuarios que el propio usuario ha destacado
Precondiciones. -
Postcondiciones. Muestra el listado de usuarios destacados por el usuario.
Escenario principal.

- El usuario solicita el listado de usuarios destacados.
- El sistema genera el listado de usuarios destacados por el usuario.

Destacar usuario

Actores. UsuarioRegistrado
Descripción. Destaca o deja de destacar a un usuario registrado.
Precondiciones. El usuario a destacar o dejar de destacar no puede ser el mismo.
Postcondiciones. Incluye al usuario indicado en la lista de destacados propios si no lo estaba o lo excluye si ya lo estaba. En caso de incluirlo si estaba bloqueado deja de estarlo.
Escenario principal.

- Ver usuarios destacados.
- El usuario solicita incluir o excluir al usuario indicado como destacado.
- El sistema incluye al usuario en la lista de destacados propios si no lo estaba o lo excluye si ya lo estaba. En caso de incluirlo si estaba bloqueado deja de estarlo.

Denunciar usuario

Actores. UsuarioRegistrado
Descripción. Denuncia a un usuario
Precondiciones. El usuario a denunciar no es él mismo.
Postcondiciones. Una denuncia es enviada al Administrador del sistema que la revisará posteriormente.
Escenario principal.

- Decide denunciar a un usuario indicando una razón.
- Prepara los datos necesarios para formalizar la denuncia.
- Denunciar.

Ver referencias dirigidas al usuario

Actores. UsuarioRegistrado
Descripción. Muestra las referencias creadas para el mismo usuario
Precondiciones. -
Postcondiciones. Muestra el listado de referencias que otros usuarios han creado para él.
Escenario principal.

- El usuario solicita el listado de referencias creadas para él mismo.
- El sistema genera el listado de referencias que otros usuarios han creado para él.

Aprobar/Desaprobar referencia

Actores. UsuarioRegistrado
Descripción. Aprueba o desaprueba referencias dirigidas al usuario
Precondiciones. -
Postcondiciones. Aprueba o desaprueba una referencia
Escenario principal.

- Ver referencias dirigidas al usuario
- El usuario selecciona una referencia pendiente e indica si la aprueba o no.
- El sistema aprueba o elimina(desaprobada) la referencia según lo indicado por el usuario.

Ver perfil usuario

Actores. UsuarioRegistrado
Descripción. Muestra toda la información pública del usuario.
Precondiciones. -
Postcondiciones. Muestra toda la información pública del usuario dependiendo de la visibilidad de los datos que haya indicado en su perfil.
Escenario principal.

- El usuario solicita ver los datos públicos de un usuario registrado.
- El sistema genera el listado de datos públicos del usuario.

Eliminar referencia

Actores. UsuarioRegistrado
Descripción. Elimina una referencia dirigida hacia otro usuario.
Precondiciones. -
Postcondiciones. La referencia queda eliminada del sistema.
Escenario principal.

- Ver perfil usuario.
- El usuario selecciona la referencia a eliminar.
- El sistema elimina la referencia.

Escribir referencia

Actores. UsuarioRegistrado
Descripción. Escribe una referencia dirigida hacia otro usuario
Precondiciones. No tiene ya una referencia dirigida a ese otro usuario.
Postcondiciones. La nueva referencia es creada, teniendo como referenciador al usuario que la escribe y como referenciado al que va dirigida. La referencia tendrá el estado pendiente de ser aprobada.
Escenario principal.

- Ver perfil usuario
- El usuario indica el tipo de vínculo entre el usuario que escribe la referencia y el usuario referenciado.
- El usuario indica una descripción.
- El sistema registra la nueva referencia.

Ver enlaces

Actores. UsuarioRegistrado
Descripción. Muestra los enlaces web del usuario.
Precondiciones. -
Postcondiciones. Muestra el listado de enlaces web del usuario.
Escenario principal.

- El usuario solicita ver sus enlaces web.
- El sistema genera el listado de enlaces web del usuario.

Añadir enlace

Actores. UsuarioRegistrado
Descripción. Añade un nuevo enlace web al perfil del usuario.
Precondiciones. -
Postcondiciones. Añade un nuevo enlace web al listado de enlaces del usuario.
Escenario principal.

- Ver enlaces
- El usuario indica el tipo de enlace web, si es un blog, perfil de twitter, facebook, etc.
- El usuario indica el enlace web.
- El usuario opcionalmente puede indicar una descripción del enlace
- El sistema añade el nuevo enlace al listado de enlaces del usuario.

Eliminar enlace

Actores. UsuarioRegistrado
Descripción. Elimina un enlace del usuario
Precondiciones. -
Postcondiciones. El enlace queda eliminado del sistema.
Escenario principal.

- Ver enlaces
- El usuario selecciona el enlace a eliminar.
- El sistema elimina el enlace.

Ver experiencia

Actores. UsuarioRegistrado
Descripción. Muestra la experiencia en empresas y/o organizaciones en las que ha participado el usuario.
Precondiciones. -
Postcondiciones. Muestra el listado de cada experiencia profesional del usuario.
Escenario principal.

- El usuario solicita ver sus experiencias profesionales.
- El sistema genera el listado de las experiencias profesionales del usuario.

Añadir experiencia

Actores. UsuarioRegistrado
Descripción. Añade una nueva experiencia profesional en el perfil del usuario.
Precondiciones. -
Postcondiciones. Añade una nueva experiencia profesional al listado de experiencias del usuario.
Escenario principal.

- Ver experiencia
- El usuario indica la organización o empresa en la que participó o trabajó.
- El usuario indica el tipo de puesto o cargo que ocupaba en la organización.
- El usuario indica las fechas de inicio y fin del periodo en el que participó.
- El usuario opcionalmente, indica una descripción.
- El sistema añade la nueva experiencia al listado de experiencias del usuario.

Eliminar experiencia

Actores. UsuarioRegistrado
Descripción. Elimina un experiencia profesional del usuario.
Precondiciones. -
Postcondiciones. La experiencia queda eliminada del sistema.
Escenario principal.

- Ver experiencia
- El usuario selecciona la experiencia a eliminar.
- El sistema elimina la experiencia.

Ver formación educativa

Actores. UsuarioRegistrado
Descripción. Muestra la formación educativa del usuario.
Precondiciones. -
Postcondiciones. Muestra el listado de formaciones educativas del usuario.
Escenario principal.

- El usuario solicita ver su formación educativa.
- El sistema genera el listado de formaciones educativas del usuario.

Añadir formación educativa

Actores. UsuarioRegistrado
Descripción. Añade una nueva formación educativa en el perfil del usuario.
Precondiciones. -
Postcondiciones. Añade una nueva formación educativa al listado de experiencias del usuario.
Escenario principal.

- Ver formación educativa
- El usuario indica el centro en el que se formó.
- El usuario indica la titulación que consiguió en ese centro.
- El usuario indica el año de inicio y fin del periodo de formación.
- El usuario opcionalmente, indica una descripción.
- El sistema añade la formación educativa al listado de formaciones del usuario.

Eliminar formación educativa

Actores. UsuarioRegistrado
Descripción. Elimina una formación educativa del usuario.
Precondiciones. -
Postcondiciones. La formación educativa queda eliminada del sistema.
Escenario principal.

- Ver formación educativa
- El usuario selecciona la formación educativa a eliminar.
- El sistema elimina la formación.

Ver méritos

Actores. UsuarioRegistrado
Descripción. Muestra los méritos o premios conseguidos por el usuario.
Precondiciones. -
Postcondiciones. Muestra el listado de méritos o premios conseguidos por el usuario.
Escenario principal.

- El usuario solicita ver sus méritos conseguidos.
- El sistema genera el listado de méritos o premios conseguidos por el usuario.

Añadir mérito

Actores. UsuarioRegistrado
Descripción. Añade un nuevo mérito en el perfil del usuario.
Precondiciones. -
Postcondiciones. Añade un nuevo mérito al listado de méritos del usuario.
Escenario principal.

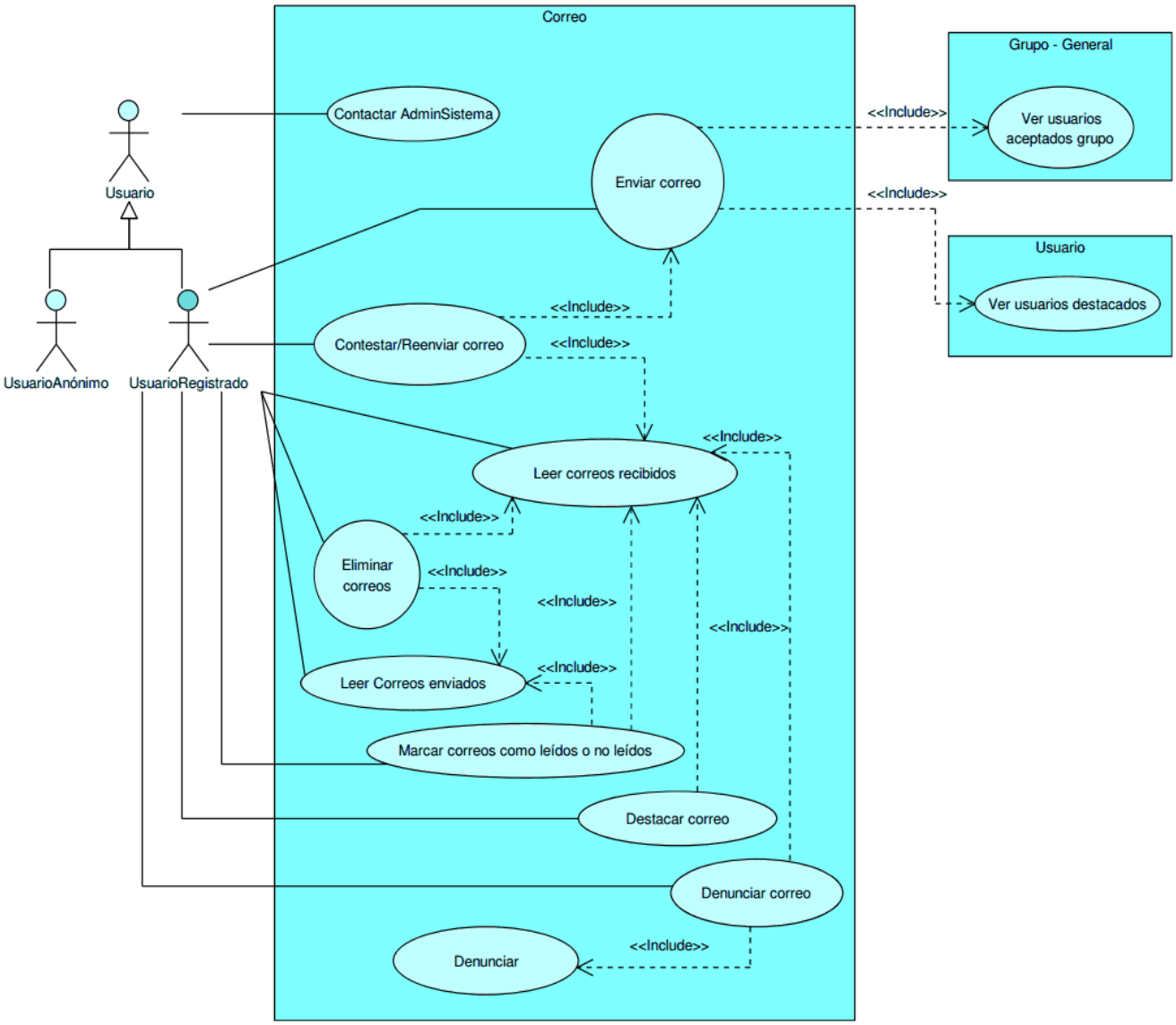
- Ver méritos
- El usuario indica el mérito o premio conseguido.
- El usuario indica la fecha en que fue otorgado.
- El sistema añade el nuevo mérito al listado de méritos del usuario.

Eliminar mérito

Actores. UsuarioRegistrado
Descripción. Elimina el mérito del usuario.
Precondiciones. -
Postcondiciones. El mérito queda eliminado del sistema.
Escenario principal.

- Ver méritos
- Selecciona un mérito a eliminar.
- El sistema elimina el mérito.

3.3.2. Casos de uso de Correo



Contactar AdminSistema

Actores. Usuario
Descripción. Envía un mensaje al Administrador del sistema indicando un tipo de mensaje, siendo los posibles, dudas, problema técnico, reclamaciones, mejoras y denuncia.
Precondiciones. -
Postcondiciones. Envía un mensaje al Administrador del sistema para posterior revisión.
Escenario principal.

- El usuario solicita los tipos de mensajes a Administrador del Sistema posibles.
- El sistema genera el listado de tipos de mensajes solicitado.
- El usuario indica el tipo de mensaje a enviar y la razón
- El sistema envía el mensaje al Administrador del sistema para posterior revisión.

Envío correo

Actores. UsuarioRegistrado
Descripción. Envía un correo a uno o más destinatarios.
Precondiciones. -
Postcondiciones. El correo es enviado a todos los destinatarios indicados.
Escenario principal.

- El usuario introduce los destinatarios, teniendo dos formas o manualmente o de un listado de usuarios. El listado se compone de los usuarios que participan en los proyectos en los que el usuario emisor participa o ha destacado, además de usuarios destacados por él mismo. Todo destinatario introducido tanto manualmente como del listado puede ser agregado como copia oculta o visible. En caso de introducirlos solo manualmente saltar al paso 4.
- Ver usuarios aceptados grupo
- Ver usuarios destacados
- El usuario escribe el asunto y el cuerpo del correo.
- El usuario envía el correo
- El sistema teniendo en cuenta que ningún destinatario tenga bloqueado al emisor enviará el correo a todos los destinatarios indicados.

Leer correos recibidos

Actores. UsuarioRegistrado
Descripción. El usuario lee los correos que le han sido enviados.
Precondiciones. -
Postcondiciones. Muestra un listado de correos donde el usuario es el destinatario
Escenario principal.

- El usuario solicita la lista de correos le han sido enviados.
- El sistema genera un listado de correos dónde el usuario es el destinatario.

Contestar/Reenviar correo

Actores. UsuarioRegistrado
Descripción. Contesta o reenvia un correo recibido.
Precondiciones. El usuario está incluido en la lista de destinatarios del correo o es el emisor de este.
Postcondiciones. El correo es contestado al emisor o reenviado a otros usuarios.
Escenario principal.

- El usuario selecciona si contestar o reenviar el correo.
- El sistema, si es contestado creará un correo listo para enviar donde el emisor aparezca como destinatario, el asunto será el mismo pero precedido de 'Re: ' y el cuerpo será el mismo del original pero añadiendo la fecha y emisor originales. En caso de ser reenviado, el destinatario y asunto aparecerán vacíos, pero en el cuerpo aparecerán tanto la fecha, emisor, destinatario, asunto y cuerpo del correo original.

Leer correos enviados

Actores. UsuarioRegistrado
Descripción. El usuario lee los correos que ha enviado.
Precondiciones. -
Postcondiciones. Muestra un listado de correos donde el usuario es el emisor.
Escenario principal.

- El usuario solicita la lista de correos que ha enviado.
- El sistema genera un listado de correos dónde el usuario es el emisor.

Eliminar correos

Actores. UsuarioRegistrado
Descripción. Elimina correos seleccionados del usuario.
Precondiciones. -
Postcondiciones. Los correos seleccionados son eliminados del sistema
Escenario principal.

- Leer correos recibidos o Leer correos enviados.
- El usuario selecciona correos o recibidos o enviados no de ambos a la vez.
- El sistema elimina los correos seleccionados.

Marcar correos como leídos o no leídos

Actores. UsuarioRegistrado
Descripción. Marca correos como leídos o no leídos.
Precondiciones. -
Postcondiciones. Los correos seleccionados son marcados como leídos o no leídos.
Escenario principal.

- Leer correos recibidos o Leer correos enviados.
- El usuario selecciona correos o recibidos o enviados no de ambos a la vez.
- El sistema marcar como leídos o no leídos los correos seleccionados.

Destacar correo

Actores. UsuarioRegistrado
Descripción. Destaca un correo recibido.
Precondiciones. -
Postcondiciones. El correo seleccionado queda destacado si no lo estaba o deja de estarlo si ya lo estaba.
Escenario principal.

- Leer correos recibidos
- El usuario selecciona un correo.
- El sistema destaca si no lo estaba o deja de destacarlo si ya lo estaba el correo seleccionado

Denunciar

Actores. Usuario
Descripción. Denuncia contenido ofensivo.
Precondiciones. -
Postcondiciones. Una denuncia es creada y enviada al administrador del sistema para posterior revisión.
Escenario principal.

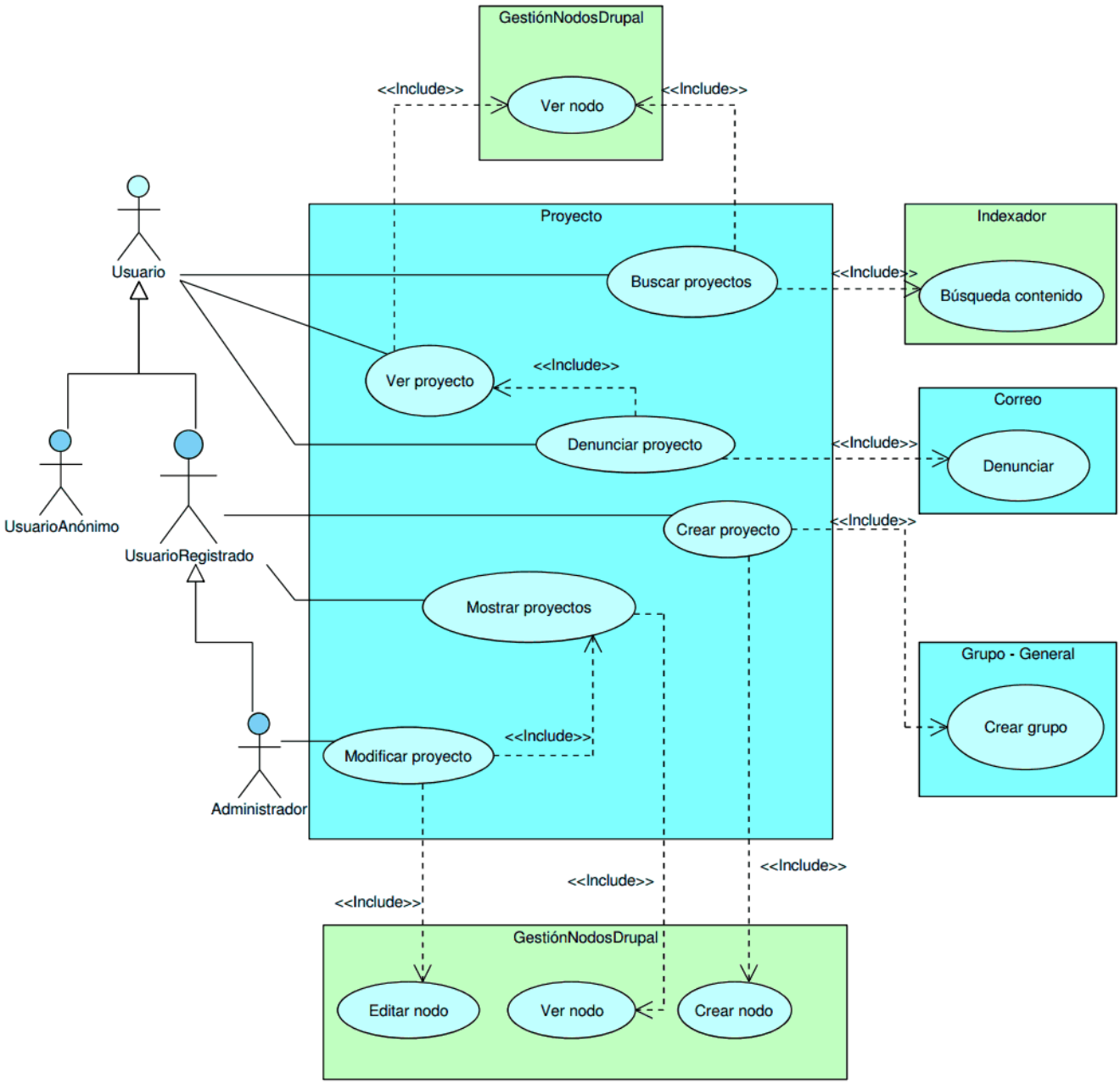
- El usuario indica los datos necesarios para la denuncia, tales como que tipo de contenido, qué contenido o usuario y la razón.
- El sistema crea y envía la denuncia al administrador del sistema.

Denunciar correo

Actores. UsuarioRegistrado
Descripción. Denuncia un correo recibido por contenido ofensivo.
Precondiciones. El usuario es uno de los destinatarios del correo.
Postcondiciones. Una denuncia es enviada al Administrador del sistema para posterior revisión.
Escenario principal.

- Leer correos recibidos
- El usuario encuentra un correo con contenido ofensivo y decide denunciarlo indicando una razón.
- El sistema prepara los datos necesarios para formalizar la denuncia.
- Denunciar

3.3.3. Casos de uso de Proyecto



Buscar proyectos

Actores. Usuario
Descripción. Realiza una búsqueda de proyectos.
Precondiciones. -
Postcondiciones. Muestra un listado de proyectos según los criterios de la búsqueda.
Escenario principal.

- El usuario introduce unos parámetros de búsqueda como el tipo de proyecto, palabras clave, además de otros filtros ofrecidos para acotar los resultados obtenidos.
- Búsqueda contenido
- El sistema muestra un listado de proyectos según los criterios de búsqueda.
- Ver nodo
- El usuario si solicita seguir aplicando filtros vuelve al paso 1 sino acaba el caso de uso

Ver proyecto

Actores. Usuario
Descripción. Muestra la información del proyecto dependiendo del tipo de vista elegida.
Precondiciones. -
Postcondiciones. En caso de elegir la vista reducida serán mostrados la imagen y descripción breve del proyecto. En cambio con la vista completa serán mostrados el vídeo o en su defecto la imagen y la descripción completa. En ambas vistas serán mostrados el título, roles o actividades solicitadas, tipo de proyecto, categoría y fecha de creación.
Escenario principal.

- El usuario indica un proyecto y el tipo de vista.
- El sistema muestra la información del proyecto dependiendo de la vista escogida.

Denunciar proyecto

Actores. Usuario
Descripción. Denuncia un proyecto por contenido ofensivo.
Precondiciones. -
Postcondiciones. Una denuncia es enviada al Administrador del sistema para posterior revisión.
Escenario principal.

- Ver proyecto
- El usuario encuentra contenido ofensivo en el proyecto y decide denunciarlo indicando una razón.
- El sistema prepara los datos necesarios para formalizar la denuncia.
- Denunciar

Crear proyecto

Actores. UsuarioRegistrado
Descripción. El usuario crea un proyecto que es publicado, de forma que sea accesible para el resto de usuarios.
Precondiciones. -
Postcondiciones. El proyecto es agregado al sistema, haciendo al creador administrador del proyecto. Además asocia este proyecto con un grupo que gestionará principalmente la asociación de los participantes, las vías de comunicación entre ellos y la financiación.
Escenario principal.

- El usuario solicita los tipos de proyectos disponibles a escoger.
- El sistema genera el listado de tipos de proyecto disponibles.
- El usuario solicita las categorías disponibles a escoger.
- El sistema genera un listado de categorías disponibles.
- El usuario introduce un título para el proyecto.
- El usuario indica el tipo de proyecto que va a crear. De tipo emprendedor o de carácter social.
- El usuario selecciona una categoría.
- El usuario introduce una descripción breve.
- El usuario introduce una descripción completa.
- El usuario introduce opcionalmente una imagen y/o un vídeo.
- El usuario indica si requerirá la descripción de aptitudes a los candidatos que envíen una solicitud de entrada.
- El usuario introduce los roles o actividades que necesitará.
- El sistema agrega el nuevo proyecto, haciendo al creado el administrador de este proyecto.
- Crear nodo
- Crear grupo

Mostrar proyectos

Actores. UsuarioRegistrado
Descripción. Muestra los proyectos según la categoría escogida.
Precondiciones. -
Postcondiciones. Muestra un listado de proyectos agrupados por membresía, en los que el usuario ha sido expulsado o destacados por él.
Escenario principal.

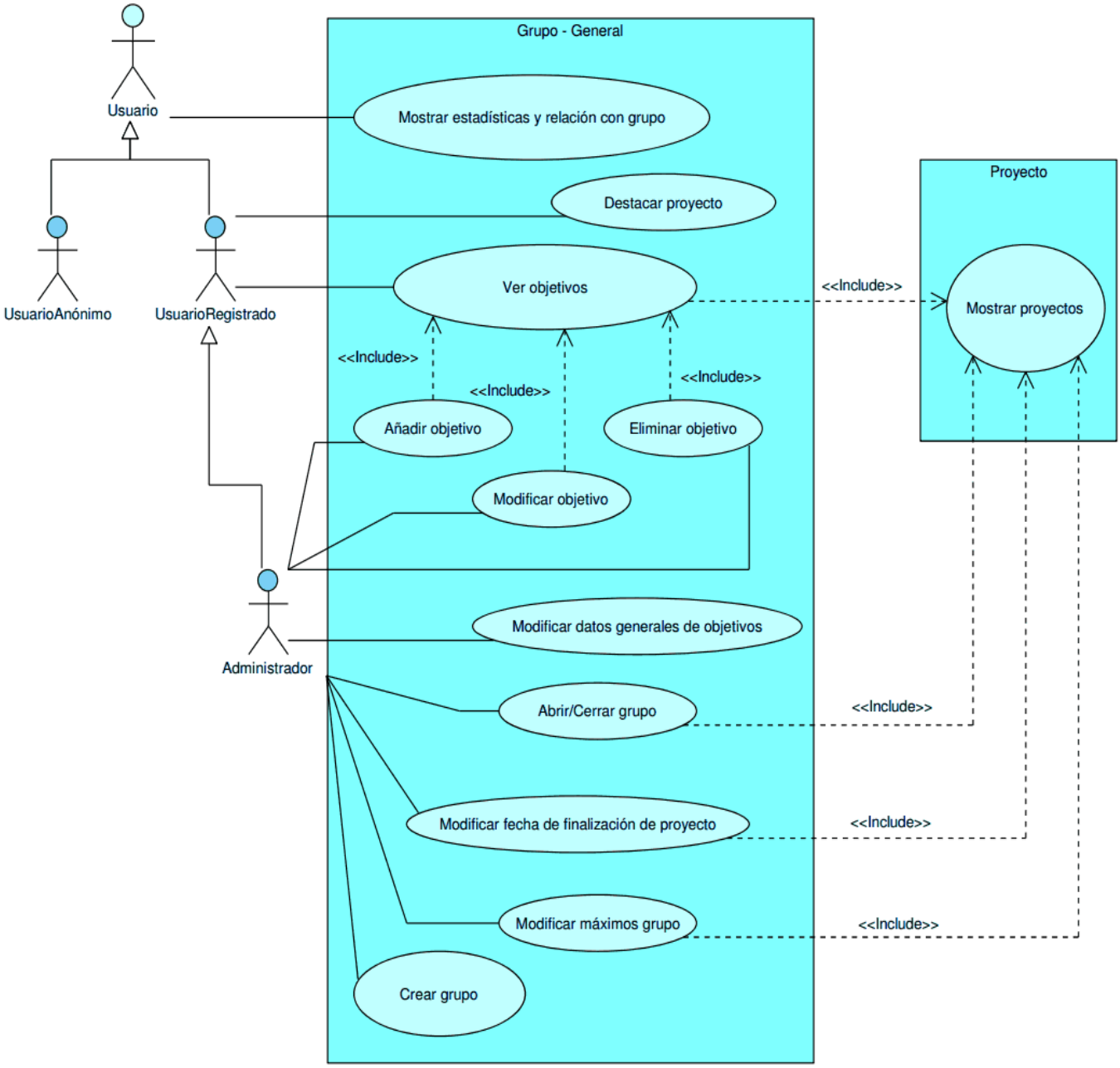
- El usuario solicita que se muestren los proyectos por un tipo de membresía, en los que ha sido expulsado o destacados por él.
- El sistema muestra el listado solicitado por el usuario.
- Ver nodo

Modificar proyecto

Actores. Administrador
Descripción. Modifica datos públicos del proyecto, excepto el tipo de proyecto el cual no se puede ya modificar.
Precondiciones. El administrador lo es del propio proyecto.
Postcondiciones. Los datos del proyecto quedan modificados.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto en el que participe como administrador y quiera modificar los datos.
- El usuario introduce los datos a modificar.
- El sistema registra los cambios realizados.
- Editar nodo

3.3.4. Casos de uso Generales de Grupo



Mostrar estadísticas y relación con el grupo

Actores. Usuario
Descripción. Muestras las estadísticas y relación con el proyecto
Precondiciones. -
Postcondiciones. Muestras estadísticas tales como porcentaje de cumplimiento de objetivo, si la recaudación está activada, lo recaudado, objetivo de recaudación, fecha prevista de finalización y estado del proyecto. Además indicad las relaciones que hay con el proyecto, tales como membresía, rol, estado del usuario en relación con el proyecto(aceptado, expulsado,...), si el usuario ha destacado el proyecto y cuantos otros usuarios lo han destacado.
Escenario principal.

- El usuario indica un proyecto del que ver las estadísticas.
- El sistema muestra las estadísticas y datos solicitados.

Destacar proyecto

Actores. UsuarioRegistrado
Descripción. Destaca o deja de destacar un proyecto por parte del usuario
Precondiciones. -
Postcondiciones. Destaca el proyecto si el usuario no lo había destacado o lo de deja de destacar si ya lo estaba.
Escenario principal.

- El usuario selecciona un proyecto.
- El sistema destaca el proyecto si el usuario no lo había destacado o lo de deja de destacar si ya lo estaba.

Ver objetivos

Actores. UsuarioRegistrado
Descripción. Muestra los objetivos del grupo
Precondiciones. -
Postcondiciones. Genera un listado con los objetivos y sus porcentajes, además del porcentaje de cumplimiento alcanzado en total.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto del que ver los objetivos.
- El sistema genera un listado con los objetivos y sus porcentajes, además del porcentaje de cumplimiento alcanzado en total.

Añadir objetivo

Actores. Administrador
Descripción. Añade un nuevo objetivo al proyecto.
Precondiciones. El administrador lo es del proyecto. No hay ningún otro objetivo con el mismo porcentaje a introducir y este es un entero entre 0 y 100.
Postcondiciones. El objetivo es agregado al listado de objetivos del proyecto.
Escenario principal.

- Ver objetivos
- El usuario añade un nuevo objetivo indicando porcentaje y descripción, teniendo en cuenta que no haya ningún otro objetivo con el mismo porcentaje.
- El sistema agrega el nuevo objetivo al listado de objetivos del proyecto.

Modificar objetivo

Actores. Administrador
Descripción. Modifica los datos de un objetivo.
Precondiciones. El administrador lo es del proyecto. No hay ningún otro objetivo con el mismo porcentaje a introducir y este es un entero entre 0 y 100.
Postcondiciones. Modifica los datos del objetivo que haya indicado el administrador.
Escenario principal.

- Ver objetivos
- El usuario indica los datos a modificar del objetivo seleccionado.
- El sistema registra los cambios realizados en el objetivo.

Eliminar objetivo

Actores. Administrador
Descripción. Elimina un objetivo del proyecto
Precondiciones. El administrador lo es del proyecto.
Postcondiciones. Elimina el objetivo del sistema.
Escenario principal.

- Ver objetivos
- El usuario selecciona un objetivo a eliminar.
- El sistema elimina el objetivo del sistema.

Modificar datos generales de objetivos

Actores. Administrador
Descripción. Modifica la visibilidad de objetivos y el porcentaje total de cumplimiento.
Precondiciones. El administrador lo es del propio proyecto.
Postcondiciones. Los datos de visibilidad y/o porcentaje total de cumplimiento son modificados.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona el proyecto al cual hará las modificaciones.
- El sistema registra los cambios en los datos de visibilidad y/o porcentajes de cumplimiento.

Abrir/Cerrar grupo

Actores. Administrador
Descripción. Abre o cierra un grupo a nuevas solicitudes.
Precondiciones. El grupo a abrir o cerrar pertenece al administrador.
Postcondiciones. El grupo queda abierto a nuevas solicitudes si estaba cerrado o es cerrado si ya estaba abierto.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto de los que es administrador y quiere modificar el estado.
- El sistema abre o cierra a nuevas solicitudes el grupo dependiendo de lo indicado por el administrador.

Modificar fecha de finalización de proyecto

Actores. Administrador
Descripción. Modifica la fecha prevista de finalización.
Precondiciones. El administrado lo es del proyecto.
Postcondiciones. La fecha de finalización queda modificada.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto de los que es administrado y quiere modificar la fecha de finalización.
- El sistema registra los cambios en la fecha prevista de finalización.

Modificar máximos de grupo

Actores. Administrador
Descripción. Modifica el máximo número de candidatos, invitados y/o miembros del proyecto que pueden participar a la vez.
Precondiciones. El administrado lo es del proyecto.
Postcondiciones. El máximo número de candidatos, invitados y/o miembros del proyecto son modificados.
Escenario principal.

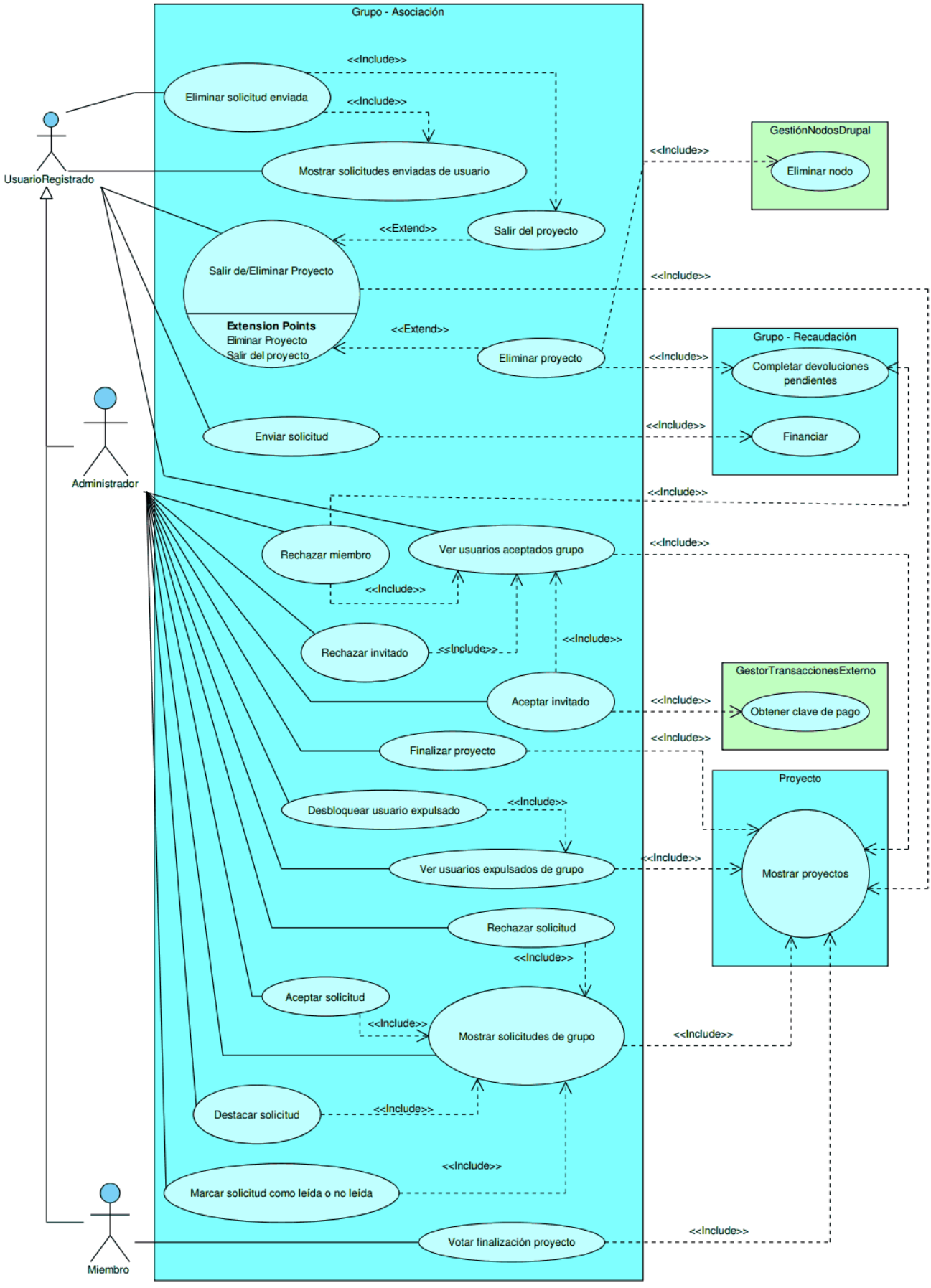
- Mostrar proyectos
- El usuario selecciona un proyecto de los que es administrado y quiere modificar los máximos.
- El sistema registra los cambios. Siempre y cuando estos no sobrepasen 40 para miembros, 60 para invitados y 100 para candidatos. Para superarlos tendría que solicitarlo al administrador del sistema.

Crear grupo

Actores. Administrador
Descripción. Crea el grupo asociado a un proyecto.
Precondiciones. No hay ningún grupo asociado previamente al proyecto. El administrador es el creador de este.
Postcondiciones. El grupo es creado y asociado al proyecto.
Escenario principal.

- El usuario indica el proyecto al que asociar con el grupo y si será requerido a los candidatos que envíen una solicitud que describan sus aptitudes.
- El sistema crea el grupo lo asocia al proyecto. Además un chat y un muro de mensajes son asociados al grupo. Por último, el administrador será el creador del grupo de la misma forma que del proyecto.

3.3.5. Casos de uso Asociación de Grupo



Mostrar solicitudes enviadas de usuario

Actores. UsuarioRegistrado
Descripción. Muestra las solicitudes enviadas por el usuario como candidato a los proyectos.
Precondiciones. -
Postcondiciones. Muestra el listado de solicitudes del usuario como candidato.
Escenario principal.

- El usuario indica el tipo de solicitudes que quiere visualizar, pendientes de aprobación, aceptadas, rechazadas o todas.
- El sistema genera el listado de solicitudes del usuario de proyectos donde es candidato.

Salir/Eliminar proyecto

Actores. UsuarioRegistrado
Descripción. El usuario sale del proyecto dejando de participar en este.
Precondiciones. El usuario es candidato, invitado, miembro o administrador del proyecto.
Postcondiciones. En caso de que el usuario sea administrador del proyecto, este queda eliminado junto con el grupo asociado y el resto de usuarios no podrán volver a entrar en este. El proyecto se eliminará siempre y cuando no hayan devoluciones o recompensas pendientes sino quedará pendiente de eliminación hasta que no se den esas condiciones. En caso de ser candidato, invitado o miembro sale del proyecto.
Principal.

- Mostrar proyectos
- Selecciona un proyecto de la lista donde participe como administrador, miembro, invitado o candidato.
- Si el usuario es administrador del proyecto, *Eliminar proyecto*
- Si el usuario es miembro, invitado o candidato, *Salir del proyecto*

Salir del proyecto

Actores. UsuarioRegistrado
Descripción. El usuario sale del proyecto dejando de participar en este.
Precondiciones. El usuario es candidato, invitado o miembro del proyecto.
Postcondiciones. El usuario sale del proyecto dejando de tener acceso a las vías de comunicación de este, como cualquier otro usuario registrado. Pero en cualquier momento puede volver a solicitar la entrada.
Extensión salir de Salir/Eliminar proyecto.

- El usuario indica que quiere salir del proyecto.
- El sistema saca al usuario del proyecto y por tanto, no tendrá acceso a las vías de comunicación que tuviera con este. En caso de ser candidato o invitado, sus solicitudes actuales de entrada también serán eliminadas.

Eliminar solicitud enviada

Actores. UsuarioRegistrado
Descripción. Elimina una solicitud enviada por él mismo a algún proyecto.
Precondiciones. La solicitud pertenece al candidato.
Postcondiciones. La solicitud es eliminada del sistema. En caso de estar pendiente de aprobación sale del proyecto como candidato.
Escenario principal.

- Ver solicitudes enviadas de usuario
- El usuario selecciona una solicitud a eliminar.
- El sistema elimina la solicitud. En caso de que estuviera pendiente de aprobación va al siguiente paso sino acaba el caso de uso.
- Salir del proyecto

Eliminar proyecto

Actores. Administrador
Descripción. Elimina el proyecto del sistema.
Precondiciones. El usuario es administrador del proyecto.
Postcondiciones. Elimina el proyecto en caso de darse las condiciones adecuadas. Si existen recompensas solicitadas, pagos completados de hace menos de 60 días y/o devoluciones pendientes, el proyecto pasará al estado Pendiente de eliminación hasta que se resolvieran esos supuestos.
Extensión salir de Salir/Eliminar proyecto.

- El administrador indica que quiere eliminar el proyecto.
- El sistema cambia el estado del proyecto a pendiente de eliminación en caso de que existan recompensas y/o devoluciones pendientes. También en caso de que hayan pagos completados realizados como máximo dentro del período que establece el gestor de transacciones. Si no se dan estas condiciones el proyecto pasa a estado Eliminado.
- El sistema expulsa automáticamente a todos los candidatos e invitados.
- El sistema en caso de que el proyecto esté en estado Eliminado, expulsará miembros y eliminar el chat, muro de mensajes y eventos del proyecto, además de todas las asociaciones con otros elementos del sistema. Si no estuviera en estado Eliminado, acaba el caso de uso sino salta al siguiente paso.
- Eliminar nodo

Enviar solicitud

Actores. UsuarioRegistrado
Descripción. El usuario solicita ingresar en el proyecto.
Precondiciones. El usuario no es ya administrador, miembro, invitado o candidato del grupo. Ni tampoco está expulsado de este. El grupo no ha llegado al máximo de candidatos. El usuario no está bloqueado por el administrador del proyecto. El grupo está abierto a nuevas solicitudes.
Postcondiciones. El usuario se convierte en candidato del grupo creando una solicitud.
Escenario principal.

- El usuario Solicita el tipo de roles que el proyecto solicita.
- El sistema genera el listado de roles del proyecto disponibles.
- El usuario indica el rol que desempeñará.
- Si el proyecto requiere la descripción de aptitudes del candidato sigue al siguiente paso o sino salta al 6.
- Introduce una descripción de sus aptitudes.

- Si el rol indicado es de tipo financiador o fuera cualquier otro tipo pero el usuario ha indicado que va a realizar una aportación inicial sigue al siguiente paso o sino salta al 8.
- Financiar
- El sistema crea la solicitud y la asocia al nuevo candidato.

Ver usuarios aceptados grupo

Actores. UsuarioRegistrado
Descripción. Muestra el listado de usuarios aceptados del grupo.
Precondiciones. -
Postcondiciones. Muestra el listado de usuarios aceptados del grupo
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto y un tipo de membresía de las que tiene permiso a ver el listado.
- El sistema genera el listado de usuarios aceptados de un grupo con la membresía indicada.

Rechazar miembro

Actores. Administrador
Descripción. Expulsa un miembro del grupo manteniéndolo así durante un máximo de 4 meses.
Precondiciones. El administrador lo es del grupo al que pertenece el miembro a expulsar.
Postcondiciones. El miembro es expulsado del grupo y tendrá el mismo acceso que cualquier otro usuario registrado que no participe en el proyecto. Aunque si realizó pagos al proyecto en un periodo que no supere el establecido por el gestor de transacciones se procedería a realizar devoluciones de los mismos y el miembro quedaría pendiente de expulsión y por tanto, aun tendría los mismos permisos que un miembro aceptado.
Escenario principal.

- Ver usuarios aceptados del grupo
- El administrador selecciona el miembro a expulsar.
- El sistema expulsa al miembro del grupo y tendrá el mismo acceso que cualquier otro usuario registrado que no participe en este. Aunque si financió el proyecto en un periodo que no supere el establecido por el gestor de transacciones se procedería a realizar devoluciones de los mismos y el miembro quedaría pendiente de expulsión y por tanto, aun tendría los mismos permisos que un miembro aceptado. En caso de que hubieran devoluciones pendientes pasa al siguiente paso sino acaba el caso de uso.
- Completar devoluciones pendientes

Rechazar invitado

Actores. Administrador
Descripción. Expulsa un invitado del grupo manteniéndolo así durante un máximo de 4 meses.
Precondiciones. El administrador lo es del grupo al que pertenece el invitado a expulsar.
Postcondiciones. El invitado es expulsado del grupo y tendrá el mismo acceso que cualquier otro usuario registrado que no participe en el proyecto.
Escenario principal.

- Ver usuarios aceptados del grupo
- El administrador selecciona el invitado a expulsar.
- El sistema expulsa al invitado del grupo y este tendrá el mismo acceso que cualquier otro usuario registrado que no participe en el proyecto.

Aceptar invitado

Actores. Administrador
Descripción. Acepta a un invitado como miembro del grupo
Precondiciones. El administrador lo es del grupo al que pertenece el invitado a expulsar. El número total de miembros actuales no llega al máximo de miembros y el usuario no está bloqueado por el propio administrador del proyecto.
Postcondiciones. El invitado pasa a ser miembro del grupo aumentando así los permisos dentro de este. En caso de tener un pago pendiente deberá completarlo antes de poder ser miembro con todos los permisos que conlleva.
Escenario principal.

- Ver usuarios aceptados del grupo
- El administrador selecciona el invitado a aceptar como miembro.
- El sistema promueve al invitado a miembro. En caso de tener un pago pendiente por haber indicado previamente una financiación inicial el nuevo miembro tendrá el estado pendiente de aceptación y tendrá que completar el pago antes de poder ser miembro aceptado con todos los permisos que conlleva.

En caso de tener el pago pendiente mencionado pasa al siguiente paso sino acaba el caso de uso.

- Obtener clave de pago

Finalizar proyecto

Actores. Administrador
Descripción. Finaliza el proyecto para así indicar que todos los objetivos de este han sido cumplidos.
Precondiciones. El administrador lo es del proyecto. El proyecto está en estado abierto, cerrado o pendiente de finalización.
Postcondiciones. El proyecto queda en estado pendiente de finalizar hasta que todos los miembros de este, en una votación unánime, indiquen que ha finalizado, pero si no hubieran miembros pasa directamente a estado finalizado. En cambio si ya estaba pendiente de finalización, queda interrumpido y vuelve a estar en estado abierto.
Escenario principal.

- Mostrar proyectos
- El administrador solicita finalizar el proyecto
- El sistema en caso de que el grupo esté en estado abierto o cerrado y hayan miembros, cambia el estado a pendiente de finalización, si no hubieran miembros directamente pasa a finalizado. En caso de ya estar pendiente de finalización interrumpe el proceso y cambia el estado a abierto.

Ver usuarios expulsados de grupo

Actores. Administrador
Descripción. Muestra el listado de usuarios expulsados del grupo.
Precondiciones. El administrador lo es del proyecto.
Postcondiciones. Muestra el listado de usuarios expulsados del grupo.
Escenario principal.

- Mostrar proyectos
- El administrador selecciona un proyecto y un tipo de membresía ya sea 'Candidato', 'Invitado' o 'Miembro'
- El sistema genera el listado de usuarios expulsados del grupo con la membresía indicada.

Desbloquear usuario expulsado

Actores. Administrador
Descripción. Desbloquea un usuario expulsado del proyecto.
Precondiciones. El administrador lo es del proyecto al que pertenece el usuario expulsado a desbloquear.
Postcondiciones. El usuario es desbloqueado y ya puede solicitar de nuevo participar en el proyecto. En caso de haber sido un candidato expulsado y existiera aun la solicitud, vuelve a ser aceptado como candidato automáticamente.
Escenario principal.

- Ver usuarios expulsados de grupo
- El administrador selecciona el usuario a desbloquear.
- El sistema desbloquea al usuario. En caso de haber sido un candidato expulsado y existiera aun la solicitud, vuelve a ser aceptado como candidato automáticamente.

Mostrar solicitudes de grupo

Actores. Administrador
Descripción. Muestra las solicitudes de candidatos del proyecto.
Precondiciones. El administrador lo es del proyecto.
Postcondiciones. Muestra el listado de solicitudes del grupo.
Escenario principal.

- Mostrar proyectos
- El administrador selecciona un proyecto e indica el tipo de solicitudes que quiere visualizar ya sean las pendientes de aceptar, las aceptadas, rechazadas o todas.
- El sistema genera el listado solicitudes del grupo.

Rechazar solicitud

Actores. Administrador
Descripción. Rechaza la solicitud de un candidato.
Precondiciones. El administrador lo es del proyecto al que pertenece el candidato que ha enviado la solicitud. La solicitud está pendiente de aceptación.
Postcondiciones. El candidato es expulsado del grupo y la solicitud rechazada. A partir de ese momento el candidato va a tener el mismo tipo de acceso que cualquier otro usuario registrado al proyecto.
Escenario principal.

- Mostrar solicitudes de grupo
- El administrador selecciona una solicitud a rechazar.
- El sistema expulsa al candidato y rechaza la solicitud.

Aceptar solicitud

Actores. Administrador
Descripción. Acepta la solicitud de un candidato y lo promueve a invitado.
Precondiciones. El administrador lo es del proyecto al que pertenece el candidato que ha enviado la solicitud. La solicitud está pendiente de aceptación.
Postcondiciones. El candidato es promovido a invitado del grupo y la solicitud es aceptada.
Escenario principal.

- Mostrar solicitudes de grupo
- El administrador selecciona una solicitud a aceptar.
- El sistema promueve al candidato a invitado y la solicitud es aceptada.

Destacar solicitud

Actores. Administrador
Descripción. Destaca o deja de destacar una solicitud.
Precondiciones. El administrador lo es del proyecto al que pertenece la solicitud.
Postcondiciones. Destaca una solicitud no destacada o deja de destacarla si ya lo estaba.
Escenario principal.

- Mostrar solicitudes de grupo
- El administrador selecciona una solicitud.
- El sistema destaca la solicitud si no lo estaba o deja de destacarla si ya lo estaba.

Marcar solicitud como leída o no leída

Actores. Administrador
Descripción. Marca como leída o no leída una solicitud.
Precondiciones. El administrador lo es del proyecto al que pertenece la solicitud.
Postcondiciones. Marca una solicitud como leída si no lo estaba o como no leída si ya lo estaba.
Escenario principal.

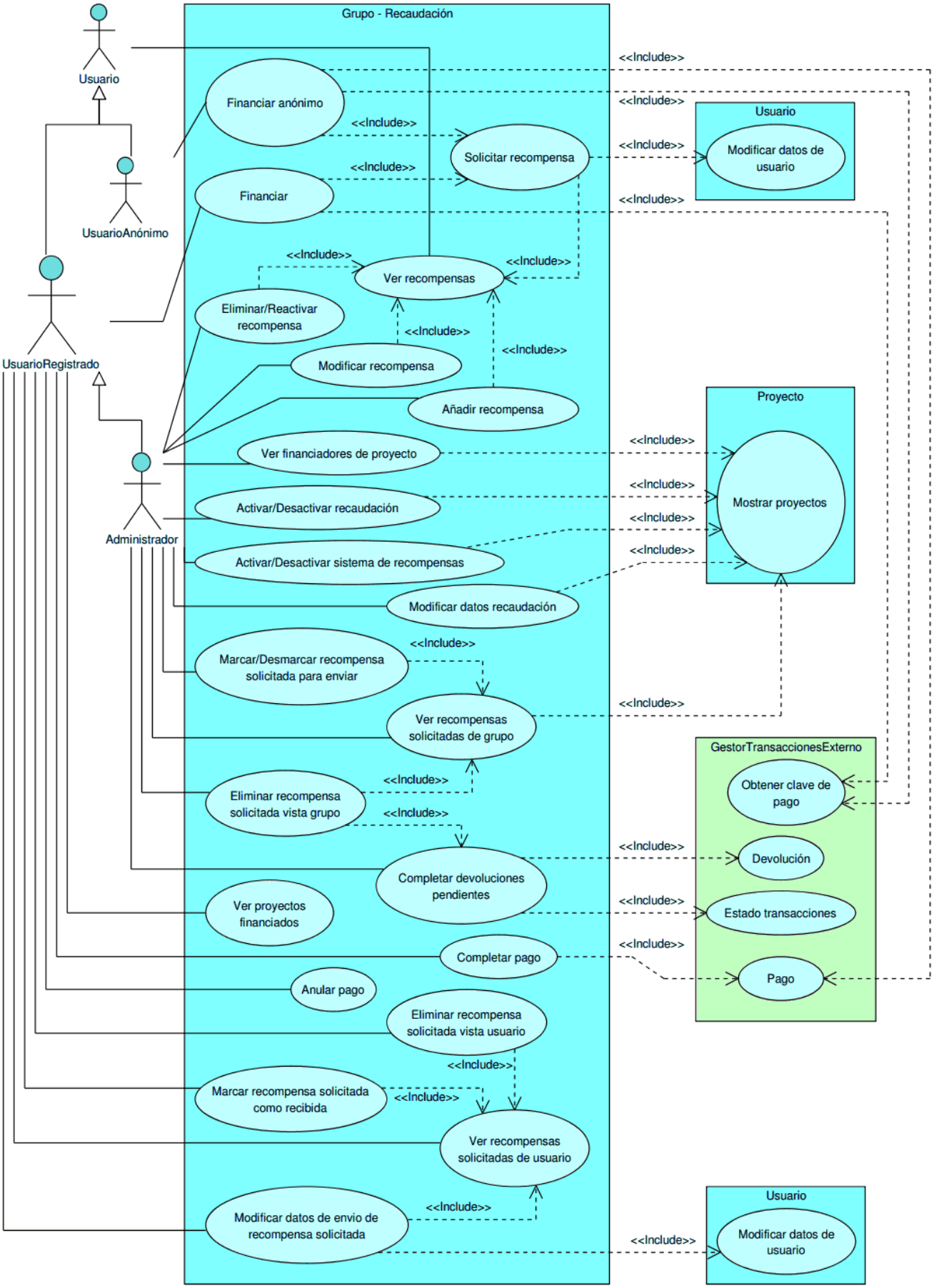
- Mostrar solicitudes de grupo
- El administrador selecciona una solicitud.
- El sistema marca la solicitud como leída si no lo estaba o como no leída si ya lo estaba.

Votar finalización proyecto

Actores. Miembro
Descripción. Un voto es emitido indicando si está de acuerdo con la finalización del proyecto.
Precondiciones. El miembro lo es del mismo proyecto por el que se realiza la votación.
Postcondiciones. El voto del miembro es guardado.
Escenario principal.

- Mostrar proyectos.
- El administrador selecciona un proyecto en el que votar.
- El sistema registra el voto emitido por el miembro.

3.3.6. Casos de uso de Financiación de Grupo



Ver recompensas

Actores. Usuario
Descripción. Muestra las recompensas disponibles para un proyecto.
Precondiciones. La recaudación y el sistema de recompensas del proyecto están activados.
Postcondiciones. Muestra el listado de recompensas disponibles para el proyecto
Escenario principal.

- El usuario indica de qué proyecto solicita el listado de recompensas.
- El sistema genera el listado de recompensas.

Solicitar recompensa

Actores. Usuario
Descripción. Solicita una recompensa por haber financiado el proyecto.
Precondiciones. El usuario ha introducido previamente una aportación con la que financiar el proyecto. La recaudación y el sistema de recompensas del proyecto están activados.
Postcondiciones. Crea una recompensa solicitada y es asociada al pago y al usuario.
Escenario principal.

- Ver recompensas
- El usuario elige del listado de recompensas mostrado una recompensa activada, que tenga una meta superior o igual a la cantidad de la aportación. Además el número de recompensas solicitadas de esta es menor que su límite.
- El sistema asocia la recompensa solicitada al usuario y al grupo del proyecto. Además actualiza los valores necesarios de la recompensa tales como el número de solicitadas.
- El usuario indica el origen de los datos si cogerlos del perfil o indicar nuevos.
- El sistema incorpora los datos y su origen en la recompensa solicitada. En caso de que los datos sean nuevos acaba el caso de uso, en cambio si su origen es el perfil del usuario, si este es registrado, va al siguiente paso.
- Modificar datos usuario

Financiar anónimo

Actores. UsuarioAnónimo
Descripción. El usuario anónimo financia el proyecto, solicitando además, si así lo indica, una recompensa.
Precondiciones. La recaudación del proyecto está activada.
Postcondiciones. Crea un pago pendiente con la cantidad indicada por el usuario y que completará inmediatamente dando los datos necesarios al gestor de transacciones.
Escenario principal.

- El usuario indica la cantidad con la que financiar un proyecto. Si quiere solicitar una recompensa va al siguiente paso sino salta al paso 3.
- Solicitar recompensa
- El sistema genera un pago pendiente y abre la pasarela de pago.
- Obtener clave de pago
- El usuario introduce email y contraseña del gestor de transacciones para completar el pago.
- Pago

Financiar

Actores. UsuarioRegistrado
Descripción. El usuario registrado financia el proyecto, solicitando además, si así lo indica, una recompensa.
Precondiciones. La recaudación del proyecto está activada.
Postcondiciones. Si el usuario no participa en el proyecto o participa como miembro o administrador,creará un pago pendiente que podrá completar más adelante. En caso de participar como candidato o invitado solo anotará el pago pendiente.
Escenario principal.

- El usuario indica la cantidad con la que financiar un proyecto.
- El sistema comprueba si hay algún pago pendiente, si es así, preguntará al usuario si quiere acumularlo junto con la nueva cantidad o desea realizar una aportación nueva. Esta opción está disponible para todos menos para el invitado del grupo.
- El usuario indica si acumula la cantidad anterior a la nueva o solo aportará la nueva. Si el usuario no participa en el proyecto o es miembro o administrador de este va al siguiente paso sino salta al 5.
- Obtener clave de pago
- El usuario si quiere solicitar una recompensa va al siguiente paso sino salta al paso 7.
- Solicitar recompensa
- El sistema genera un pago pendiente en caso de que el usuario no participe en el proyecto o sea miembro o administrador de este. En cambio si el usuario es candidato o invitado del proyecto, simplemente quedará anotada la cantidad pendiente de pago. El pago pendiente será generado en el momento en que llegue a ser miembro.

Eliminar/Reactivar recompensa

Actores. Administrador
Descripción. Elimina una recompensa de un proyecto que estuviera activada o la reactiva si estaba desactivada.
Precondiciones. El administrador lo es del proyecto. La recaudación y el sistema de recompensas del proyecto están activados.
Postcondiciones. Elimina la recompensa del sistema en caso de no haber recompensas solicitadas para esta. En caso de haberlas, se desactiva. Si la recompensa estaba ya desactivada entonces es reactivada.
Escenario principal.

- Ver recompensas
- El usuario selecciona la recompensa a eliminar.
- El sistema elimina la recompensa del sistema en caso de no haber recompensas solicitadas para esta. En caso de haberlas, se desactiva. Si la recompensa estaba ya desactivada entonces es reactivada.

Modificar recompensa

Actores. Administrador
Descripción. Modifica los datos de la recompensa.
Precondiciones. El administrador, lo es del proyecto. La recaudación y el sistema de recompensas del proyecto están activados. La meta y la estimación son mayor o igual que 1. No existe ninguna otra recompensa del proyecto con la misma meta que la que va a ser introducida.
Postcondiciones. Modifica los datos de la recompensa que el administrador haya indicado.
Escenario principal.

- Ver recompensas
- El usuario indica los datos a modificar de la recompensa seleccionada.
- El sistema registra los cambios en la recompensa.

Añadir recompensa

Actores. Administrador
Descripción. Crea una nueva recompensa asociada a un proyecto.
Precondiciones. El administrador, lo es del proyecto donde se creará la nueva recompensa. La recaudación y el sistema de recompensas del proyecto están activados.
Postcondiciones. Asocia la nueva recompensa al grupo seleccionado.
Escenario principal.

- Ver recompensas
- El usuario indica la meta, teniendo en cuenta que no haya ninguna otra recompensa con la misma meta.
- El usuario introduce el nombre de la recompensa.
- El usuario introduce la descripción de la recompensa.
- El usuario introduce el límite en el número de recompensas solicitada que pueden existir a la vez.
- El usuario indica el número de semanas que prevé que puede tardar en estar preparada la recompensa para ser enviada desde que se completa el pago.
- El sistema asocia la nueva recompensa al grupo y por tanto, estará disponible para solicitarla a la hora de financiar el proyecto.

Ver financiadores de proyecto

Actores. Administrador
Descripción. Muestra el listado de financiadores del proyecto.
Precondiciones. El administrador, lo es del proyecto.
Postcondiciones. Muestra el listado de financiadores del proyecto junto con lo financiado, su membresía en este y su recompensa en caso de haberla solicitado.
Escenario principal.

- Mostrar proyectos
- El usuario indica el proyecto del que quiere ver el listado de financiadores.
- El sistema genera el listado de financiadores del proyecto junto con lo financiado, su membresía en este y su recompensa en caso de haberla solicitado.

Activar/Desactivar recaudación

Actores. Administrador
Descripción. Activa o desactiva la recaudación del proyecto.
Precondiciones. El administrador, lo es del proyecto.
Postcondiciones. Activa o desactiva la recaudación del proyecto. En caso de ser desactivada elimina los pagos pendientes, menos de los miembros financiadores. Los miembros que no tiene el rol de financiador y que estuvieran en estado pendiente de aceptación pasan a estado aceptado.
Escenario principal.

- Mostrar proyectos
- El usuario solicita activar o desactivar la recaudación
- El sistema registra los cambios y realiza los cambios oportunos.

Activar/Desactivar sistema de recompensas

Actores. Administrador
Descripción. Activa o desactiva el sistema de recompensas del proyecto.
Precondiciones. El administrador, lo es del proyecto. La recaudación del proyecto está activada.
Postcondiciones. El sistema de recompensas es activado o desactivado según lo solicitado por el administrador.
Escenario principal.

- Mostrar proyectos
- El usuario solicita activar o desactivar el sistema de recompensas.
- El sistema registra los cambios y realiza los cambios oportunos.

Modificar datos recaudación

Actores. Administrador
Descripción. Realiza modificaciones en los datos de recaudación, tales como divisa, cuenta, aportación mínima y objetivo de financiación.
Precondiciones. El administrador, lo es del proyecto. La recaudación del proyecto está activada.
Postcondiciones. Realiza las modificaciones en los datos de recaudación indicados por el administrador.
Escenario principal.

- Mostrar proyectos
- El usuario introduce los nuevos datos que quiera modificar.
- El sistema registra los cambios.

Ver recompensas solicitadas de grupo

Actores. Administrador
Descripción. Muestra el listado de recompensas solicitadas por los usuarios que han financiado el proyecto.
Precondiciones. El administrador, lo es del proyecto.
Postcondiciones. Muestra el listado de recompensas solicitadas del proyecto
Escenario principal.

- Mostrar proyecto
- El usuario indica el proyecto del que quiere ver las recompensas solicitadas.
- El sistema genera el listado de recompensas solicitadas del proyecto.

Marcar/Desmarcar recompensa solicitada para enviar

Actores. Administrador
Descripción. Marca o desmarca una recompensa para ser enviada al usuario que la solicitó.
Precondiciones. El administrador, lo es del proyecto al que pertenece la recompensa solicitada. La recompensa solicitada tiene el pago completado.
Postcondiciones. La recompensa solicitada es marcada o desmarcada como preparada para ser enviada.
Escenario principal.

- Ver recompensas solicitadas de grupo
- El usuario selecciona una recompensa y la marca o desmarca como preparada para ser enviada.
- El sistema registra los cambios.

Completar devoluciones pendientes

Actores. Administrador
Descripción. Completa las devoluciones pendientes comunicándose con la empresa gestora de transacciones.
Precondiciones. El administrador, lo es del proyecto.
Postcondiciones. La recompensa solicitada es marcada o desmarcada como preparada para ser enviada.
Escenario principal.

- El administrador solicita completar las devoluciones pendientes de un proyecto.
- El sistema solicita a la empresa gestora si las devoluciones pendientes ya está completadas ejecutando Estado transacciones
- El sistema en caso de comprobar que no están completadas va al siguiente paso, sino realiza los cambios necesarios y acaba el caso de uso.
- Devolución

Eliminar recompensa solicitada vista grupo

Actores. Administrador
Descripción. Elimina una recompensa solicitada del proyecto.
Precondiciones. El administrador, lo es del proyecto al que pertenece la recompensa solicitada. La transacción de la recompensa es del tipo pago pendiente o pago completado. La recompensa solicitada tiene el estado pendiente de pago, pago completado o pendiente de envío. Si la transacción de la recompensa solicitada es del tipo Pago completado, deberá haberse realizado este pago hace menos de 60 días para poder realizar la devolución tal como establece la empresa gestora de transacciones escogida.
Postcondiciones. Elimina la recompensa solicitada del proyecto si la transacción es de tipo pago pendiente, sino la esta pasará a ser del tipo devolución y se procederá a realizar las devoluciones pendientes.
Escenario principal.

- Ver recompensas solicitadas de grupo
- El usuario selecciona una recompensa a eliminar.
- El sistema elimina la recompensa si es de pago pendiente y si es completado crea una devolución. Si la devolución es creada va al siguiente paso sino acaba el caso de uso.
- Completar devoluciones pendientes

Ver proyectos financiados

Actores. UsuarioRegistrado
Descripción. Muestra el listado de los proyectos que ha financiado.
Precondiciones. -
Postcondiciones. Muestra el listado de proyectos financiados por el usuario informando no solo del proyecto en cuestión, sino también de la cantidad, fecha y recompensa en caso de haberla solicitado.
Escenario principal.

- El usuario solicita el listado de los proyectos que ha financiado.
- El sistema genera el listado de proyectos financiados por el usuario.

Completar pago

Actores. UsuarioRegistrado
Descripción. Completa el pago pendiente de alguna aportación a un proyecto.
Precondiciones. Existe un pago pendiente del usuario con el proyecto y la recaudación está activada. El usuario no es ni candidato ni invitado del grupo.
Postcondiciones. Completa el pago pendiente, convirtiendo este pago pendiente en completado. Si tenía alguna recompensa asociada, pasará a estado pago completado. El pago pasará a formar parte de la recaudación del proyecto.
Escenario principal.

- El usuario solicita completar un pago pendiente con el proyecto
- El sistema comprueba si el pago ya se completó en algún intento anterior.
- Estado transacciones
- El sistema en caso de que efectivamente ya estuviera el pago completado realiza las actualizaciones necesarias y acaba el caso de uso, sino pasa al siguiente paso.
- El usuario indica la cuenta y contraseña que tenga con la empresa gestora de transacciones.
- Pago
- El sistema en caso de comprobar que el pago se ha completado, lo convierte de pendiente a completado de la misma forma que la recompensa, si es que tuviera una asociada. Además el pago pasará a formar parte de la recaudación del proyecto. En caso de que el usuario fuese un miembro pendiente de aceptación pasaría a estar aceptado.

Anular pago

Actores. UsuarioRegistrado
Descripción. Anula un pago pendiente con el proyecto.
Precondiciones. Existe un pago pendiente del usuario con el proyecto y el usuario no es invitado del grupo ni miembro pendiente de aceptación.
Postcondiciones. El pago pendiente es eliminado y actualiza los datos necesarios.
Escenario principal.

- El usuario solicita anular el pago pendiente con el proyecto.
- El sistema elimina el pago pendiente y actualiza los datos necesarios.

Ver recompensas solicitadas de usuario

Actores. UsuarioRegistrado
Descripción. Muestra el listado de recompensas solicitadas por el usuario.
Precondiciones. -
Postcondiciones. Muestra el listado de recompensas solicitadas por el usuario según el tipo indicado.
Escenario principal.

- El usuario solicita ver el listado de recompensas que ha solicitado e indica el tipo, ya sea pendiente de pago, de pago completado, pendiente de envío o pendiente de devolución.
- El sistema genera el listado de recompensas solicitadas según el tipo indicado.

Eliminar recompensa solicitada vista usuario

Actores. UsuarioRegistrado
Descripción. Elimina una recompensa solicitada por el mismo usuario
Precondiciones. Existe la recompensa a eliminar, es del usuario y tiene el pago pendiente o completado,
Postcondiciones. La recompensa es eliminada. Además puede eliminar el pago pendiente en caso de no ser invitado o miembro con estado pendiente de aceptación del proyecto.
Escenario principal.

- Ver recompensas solicitadas de usuario
- El usuario selecciona la recompensa a eliminar.
- El sistema da las opciones de eliminación. 'Eliminar solo recompensa' y 'Eliminar recompensa y aportación'. Esta última solo en caso de ser un pago pendiente y que no participe en el proyecto como invitado o miembro en estado pendiente de aceptación.
- El usuario selecciona la opción
- El sistema elimina la recompensa junto con la aportación si así se indica y está permitido.

Marcar recompensa solicitada como recibida

Actores. UsuarioRegistrado
Descripción. Marca una recompensa solicitada por el usuario como recibida.
Precondiciones. Existe la recompensa, es del usuario y tiene el estado pendiente de enviar.
Postcondiciones. La recompensa solicitada es eliminada del sistema.
Escenario principal.

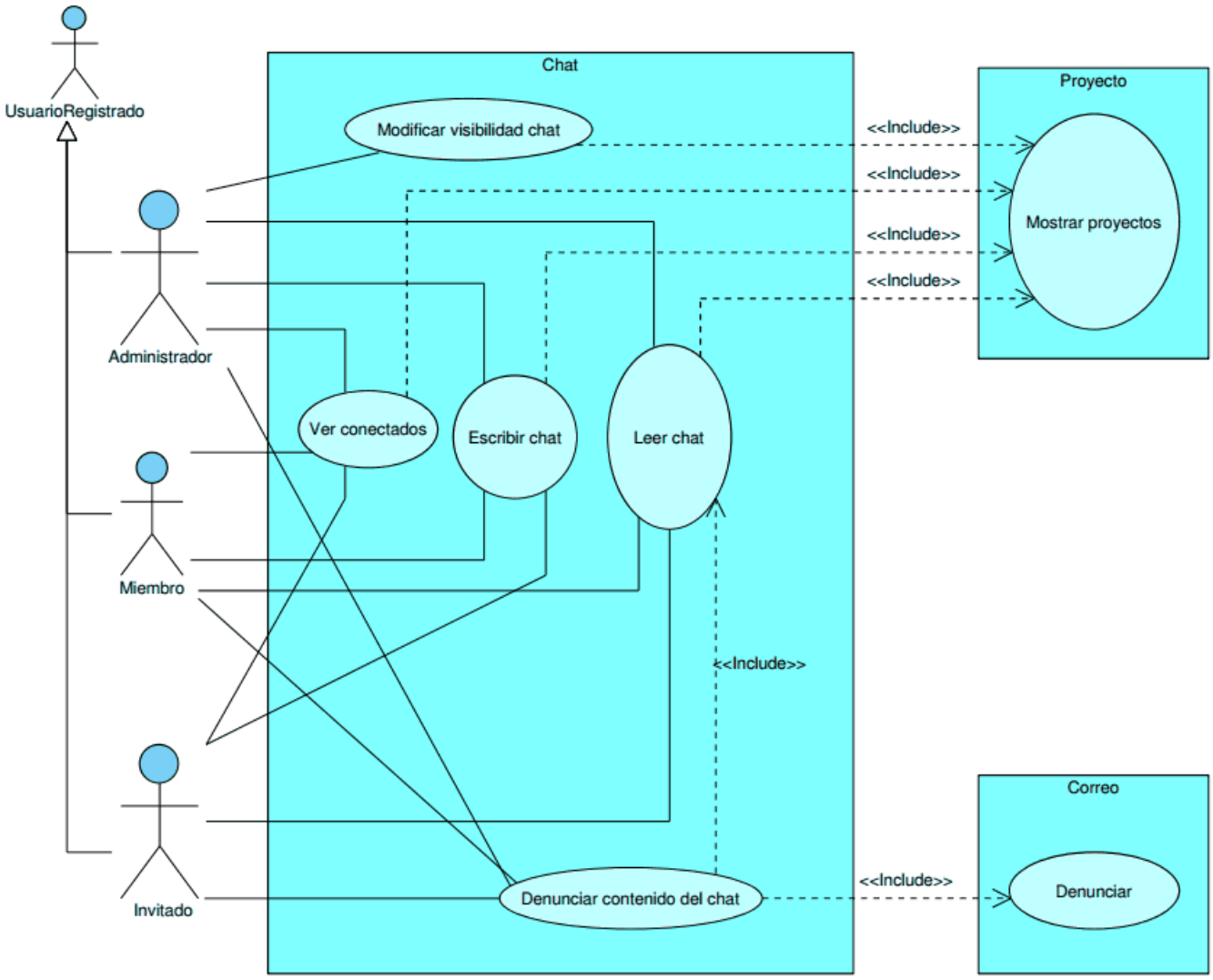
- Ver recompensas solicitadas de usuario
- Selecciona una recompensa a marcar como recibida.
- El sistema elimina la recompensa del sistema.

Modificar datos de envió de recompensa solicitada

Actores. UsuarioRegistrado
Descripción. Modifica los datos de envío de la recompensa solicitada.
Precondiciones. Existe la recompensa, es del usuario y no tiene el pago pendiente de devolución.
Postcondiciones. Modifica los datos de envío de la recompensa que el usuario haya indicado.
Escenario principal.

- Ver recompensas solicitadas de usuario
- Indica los datos a modificar, tanto el origen de estos, como los datos de envío en si.
- El sistema registra los cambios. En caso de que el origen sea la propia recompensa el caso de uso termina. En cambio si el origen de los datos es el perfil del usuario entonces va al siguiente paso.
- Modificar datos de usuario

3.3.7. Casos de uso de Chat



Modificar visibilidad chat

Actores. Administrador
Descripción. Modifica la visibilidad del chat.
Precondiciones. El administrador lo es del grupo al que pertenece el chat.
Postcondiciones. Modifica la visibilidad para que sea visible solo a administrador o miembros o administrador, miembros y invitados.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto de los que es administrador y modifica la visibilidad.
- El sistema registra los cambios.

Ver conectados

Actores. Administrador, Miembro, Invitado
Descripción. Muestra los conectados al chat en un instante de tiempo
Precondiciones. El usuario es administrador, miembro o invitado del grupo al que pertenece el chat.
Postcondiciones. Muestra el listado de conectados al chat en este instante.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto en el cual participa como administrador, miembro o invitado.
- El sistema genera el listado de usuarios conectados al chat indicando su membresía.

Leer chat

Actores. Administrador, Miembro, Invitado
Descripción. Obtiene los mensajes de chat de un proyecto.
Precondiciones. El usuario es administrador, miembro o invitado del grupo al que pertenece el chat.
Postcondiciones. El usuario se conecta al chat. Si el usuario es invitado solo obtendrá mensajes creados con visibilidad para administrador, miembros e invitados. Tanto administrador como miembros recibirán todos los mensajes sin importar la visibilidad. Los mensajes están ordenados por fecha de más antiguo a más reciente.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto del cual es administrador, miembro o invitado.
- El sistema conecta al usuario al chat y le entrega un listado de mensajes ordenados y teniendo en cuenta la visibilidad y membresía.

Escribir chat

Actores. Administrador, Miembro, Invitado
Descripción. Escribe un mensaje de chat de un proyecto.
Precondiciones. El usuario es administrador, miembro o invitado del grupo al que pertenece el chat.
Postcondiciones. El usuario se conecta al chat. Un mensaje de chat es creado y asociado con el chat del proyecto y el usuario. La visibilidad del mensaje será la misma que la del chat en el instante en que se haya creado el mensaje.
Escenario principal.

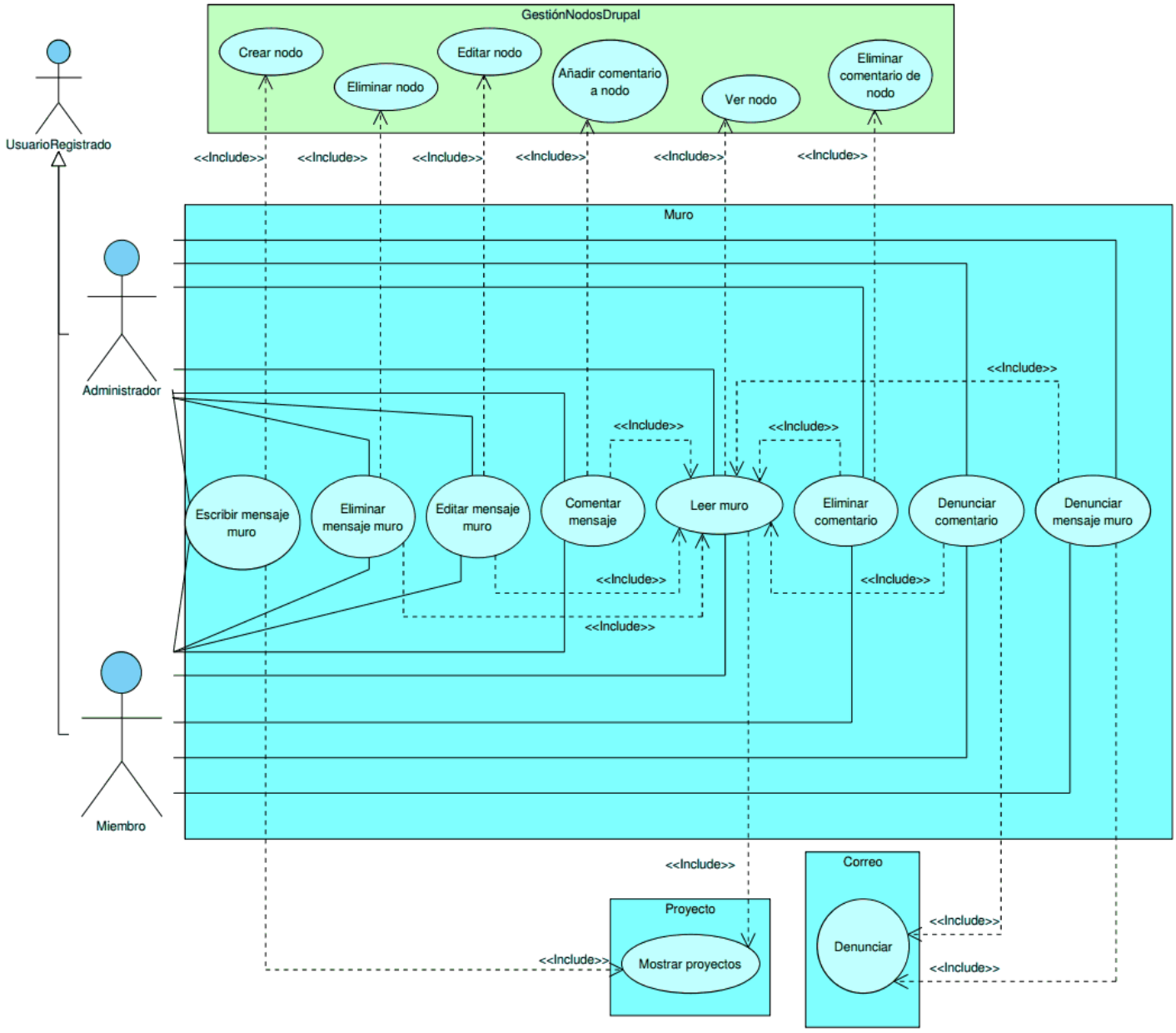
- Leer chat
- El usuario escribe el mensaje y lo envía.
- El sistema conecta al usuario al chat. El mensaje es creado y registrado. Además es entregado automáticamente al resto de usuarios conectados al mismo chat y que tienen permiso para verlo.

Denunciar chat

Actores. Administrador, Miembro, Invitado
Descripción. Denuncia la conversación visible actualmente de mensajes de chat por contenido ofensivo.
Precondiciones. El usuario es administrador, miembro o invitado del grupo al que pertenece el chat.
Postcondiciones. Una denuncia es enviada al Administrador del sistema para posterior revisión.
Escenario principal.

- Leer chat
- El usuario encuentra mensajes ofensivos y decide denunciarlo indicando una razón.
- El sistema prepara los datos necesarios para formalizar la denuncia.
- Denunciar

3.3.8. Casos de uso de Muro



Escribir mensaje muro

Actores. Administrador, Miembro
Descripción. El usuario escribe un mensaje en el muro del proyecto.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece el muro.
Postcondiciones. El mensaje queda registrado en el muro.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto en el que participe como administrador o miembro y escribe un mensaje de muro.
- El sistema registra el mensaje y lo hace visible automáticamente al resto de usuarios con permiso para ver el muro.
- Crear nodo

Leer muro

Actores. Administrador, Miembro
Descripción. Muestra los mensajes de muro de un proyecto junto con los comentarios de estos.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece el muro.
Postcondiciones. Muestra el listado de mensajes de muro ordenados cronológicamente de más recientes a más antiguos. Cada mensaje tendrá a su vez asociados los comentarios que le pertenezcan ordenados de más antiguo a más reciente.
Escenario principal.

- Mostrar proyectos
- El administrador o miembro selecciona un proyecto en el que participe como tal.
- El sistema genera el listado de mensajes de muro con sus comentarios asociados.
- Ver nodo

Eliminar mensaje muro

Actores. Administrador, Miembro
Descripción. El usuario elimina un mensaje del muro.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece el muro. Si es miembro, el mensaje deberá haber sido creado por él mismo.
Postcondiciones. El mensaje es eliminado del sistema junto con los comentarios asociados.
Escenario principal.

- Leer muro
- El usuario selecciona el mensaje a eliminar.
- El sistema elimina el mensaje junto con los comentarios asociados.
- Eliminar nodo

Editar mensaje muro

Actores. Administrador, Miembro
Descripción. El usuario edita un mensaje del muro.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece el mensaje y es el autor.
Postcondiciones. El mensaje es modificado.
Escenario principal.

- Leer muro
- El usuario selecciona el mensaje a editar.
- El usuario introduce el nuevo texto del mensaje.
- El sistema registra los cambios.
- Editar nodo

Comentar mensaje

Actores. Administrador, Miembro
Descripción. El usuario escribe un comentario a un mensaje de muro.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece el mensaje a comentar.
Postcondiciones. El comentario es asociado al mensaje de muro.
Escenario principal.

- Leer muro
- El usuario selecciona el mensaje al que añadir el comentario y lo escribe.
- El sistema asocia el comentario con el mensaje y es visible automáticamente por el resto de usuarios con permiso para leer el muro.
- Añadir comentario a nodo

Eliminar comentario

Actores. Administrador, Miembro
Descripción. Elimina un comentario de un mensaje de muro.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece el comentario a eliminar. Si es miembro, el comentario o mensaje que lo contiene deberá haber sido creado por él mismo.
Postcondiciones. El comentario es eliminado del sistema.
Escenario principal.

- Leer muro
- El usuario selecciona un mensaje de muro y abre la lista de comentarios asociados a este.
- El usuario selecciona el comentario a eliminar.
- El sistema elimina el comentario del sistema.
- Eliminar comentario de nodo

Denunciar comentario

Actores. Administrador, Miembro
Descripción. Denuncia un comentario de un mensaje de muro por contenido ofensivo.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece comentario.
Postcondiciones. Una denuncia es enviada al Administrador del sistema para posterior revisión.
Escenario principal.

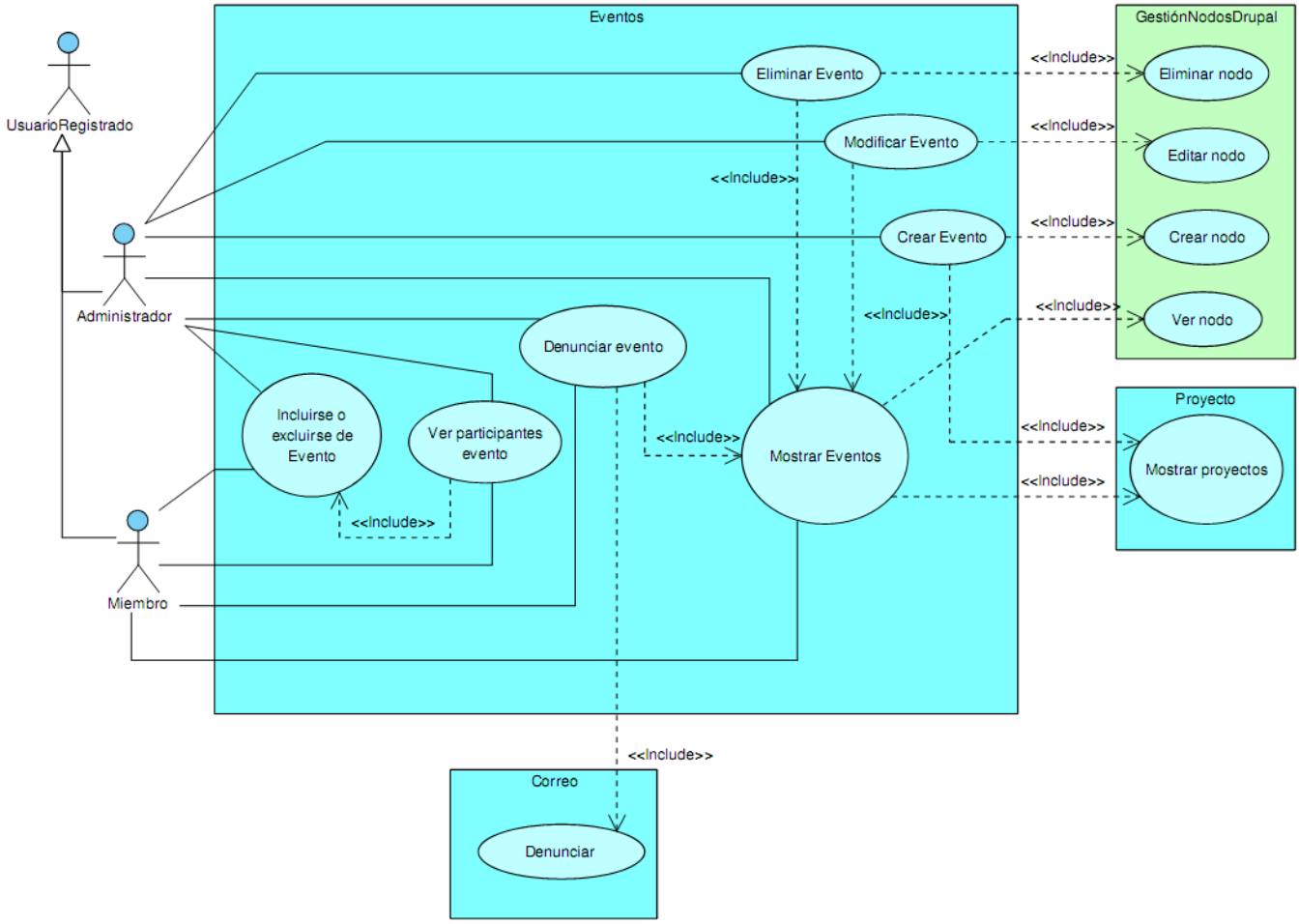
- Leer muro
- El usuario encuentra un comentario ofensivo y decide denunciarlo indicando una razón.
- El sistema prepara los datos necesarios para formalizar la denuncia.
- Denunciar

Denunciar mensaje muro

Actores. Administrador, Miembro
Descripción. Denuncia un mensaje de muro por contenido ofensivo.
Precondiciones. El usuario es administrador o miembro del grupo al que pertenece mensaje.
Postcondiciones. Una denuncia es enviada al Administrador del sistema para posterior revisión.
Escenario principal.

- Leer muro
- El usuario encuentra un mensaje de muro ofensivo y decide denunciarlo indicando una razón.
- El sistema prepara los datos necesarios para formalizar la denuncia.
- Denunciar

3.3.9. Casos de uso de Eventos



Mostrar eventos

Actores. Administrador, Miembro
Descripción. Muestra los eventos de un proyecto.
Precondiciones. El usuario es administrador o miembro del proyecto.
Postcondiciones. Muestra el listado de eventos del proyecto ordenados de más recientes a más antiguos.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto del cual es administrador o miembro.
- El sistema genera el listado de eventos del proyecto.
- Ver nodo

Ver participantes evento

Actores. Administrador, Miembro
Descripción. Muestra los eventos de un proyecto.
Precondiciones. El usuario es administrador o miembro del proyecto al que pertenece el evento.
Postcondiciones. Muestra el listado usuarios que participan en el evento.
Escenario principal.

- Mostrar eventos
- El usuario selecciona un evento.
- El sistema genera el listado de participantes del evento.

Incluirse/Excluirse de evento

Actores. Administrador, Miembro
Descripción. El usuario se incluye o excluye de participar en un evento.
Precondiciones. El usuario es administrador o miembro del proyecto al que pertenece el evento.
Postcondiciones. Si el usuario participaba en el evento queda excluido de este sino será incluido.
Escenario principal.

- Ver participantes evento
- El usuario después de comprobar si participa o no en el evento indica si se incluye o excluye de este.
- El sistema registra los cambios.

Crear evento

Actores. Administrador
Descripción. Crea un evento en un proyecto.
Precondiciones. El usuario es administrador del proyecto.
Postcondiciones. Un nuevo evento es creado con un título, lugar, fecha y descripción.
Escenario principal.

- Mostrar proyectos
- El usuario selecciona un proyecto del cual es administrador.
- El usuario indicar un título para el nuevo evento.
- El usuario indicar una fecha.
- El usuario indica el lugar donde se producirá el evento.
- El usuario indica la descripción del evento.
- El sistema crea el evento en el grupo con los datos introducidos.
- Crear nodo

Modificar evento

Actores. Administrador
Descripción. Modifica los datos de un evento.
Precondiciones. El usuario es administrador del proyecto al que pertenece el evento.
Postcondiciones. El evento es modificado según los datos indicados por el administrador.
Escenario principal.

- Mostrar eventos
- El usuario selecciona un evento a modificar.
- El usuario introduce los datos a modificar.
- El sistema registra los cambios.
- Editar nodo

Eliminar evento

Actores. Administrador
Descripción. Elimina un evento.
Precondiciones. El usuario es administrador del proyecto al que pertenece el evento.
Postcondiciones. El evento es eliminado del sistema.
Escenario principal.

- Mostrar eventos
- El administrador selecciona el evento a eliminar.
- El sistema elimina el evento.
- Eliminar nodo

Denunciar evento

Actores. Miembro
Descripción. Denuncia un evento por contenido ofensivo.
Precondiciones. El usuario es miembro del grupo al que pertenece evento.
Postcondiciones. Una denuncia es enviada al Administrador del sistema para posterior revisión.
Escenario principal.

- Mostrar eventos
- El usuario encuentra un evento con contenido ofensivo y decide denunciarlo indicando una razón.
- El sistema prepara los datos necesarios para formalizar la denuncia.
- Denunciar

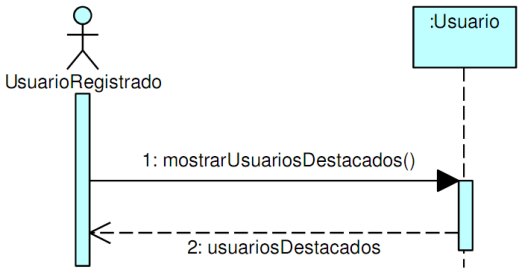
3.4. Funcionalidades principales

A continuación son mostrados los diagramas de secuencia que consideramos más importantes junto con la especificación de las operaciones. Están divididos de la misma forma que los casos de uso anteriores en subsistemas. El resto de diagramas de secuencia se encuentran en el Apéndice II – Diagramas de secuencia.

3.4.1. Diagramas de secuencia de usuario

Ver usuarios destacados

Muestra los usuarios que ha destacado el propio usuario. Además es utilizada en el momento de crear un correo ya que permite ver estos usuarios en el listado de contactos de correo.

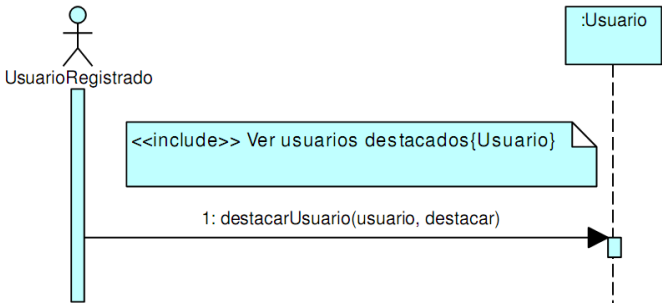


Operación: mostrarUsuariosDestacados() : Set(UsuarioRegistrado)
Descripción: Muestra los usuarios que el propio usuario ha destacado
Precondiciones: -
Salida:
Genera el listado de usuarios destacados por el usuario

let usuariosDestacados : Set(UsuarioRegistrado) = self.destacado

Destacar usuario

Destaca o deja de destacar a un usuario siempre que no sea él mismo, incluyéndolo o excluyéndolo de su listado de usuarios destacados. Además en caso de destacar al usuario y que este estuviese bloqueado por él, deja de estar bloqueado. Con lo cual, podría volver a realizar acciones que bloqueado no podía. Estas acciones incluyen enviar correos al usuario que ya no le bloquea y enviar solicitudes de entrada, siempre y cuando se cumplan otras condiciones, en los proyectos creados por el usuario que ya no le bloquea.



Operación: destacarUsuario(usuario: UsuarioRegistrado, destacar: Boolean)
Descripción: Destaca o deja de destacar a un usuario registrado
Precondiciones:

El usuario a incluir no es él mismo y no está ya incluido
if destacar then
usuario <> self and self.destacado -> excludes(usuario)
endif

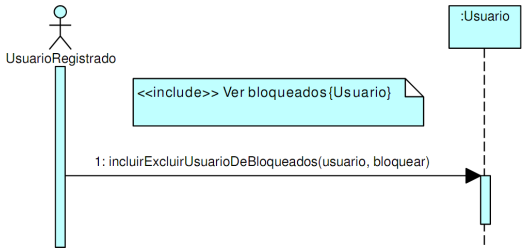
El usuario a excluir no es él mismo y ya está incluido.
if not destacar then
usuario <> self and self.destacado -> includes(usuario)
endif

Postcondiciones:
Incluye al usuario en la lista de destacados propios si no lo estaba o lo excluye si ya lo estaba.
En caso de incluirlo si estaba bloqueado deja de estarlo.

if destacar then
self.destacado-> includes(usuario)
self.bloqueado -> excludes(usuario)
else
self.destacado -> excludes(usuario)
endif

Ver usuarios bloqueados

Muestra los usuarios que ha bloqueado el propio usuario.

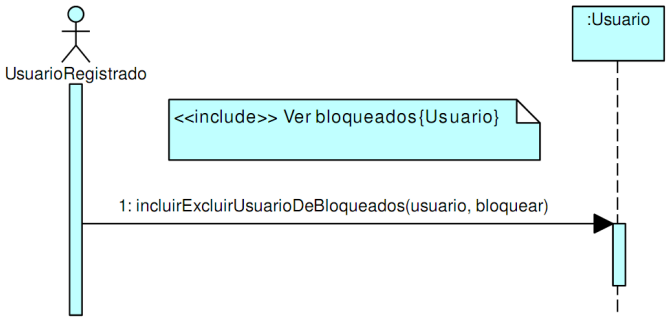


Operación: mostrarUsuariosBloqueados() : Set(UsuarioRegistrado)
Descripción: Muestra los usuarios que el propio usuario ha bloqueado
Precondiciones: -
Salida:
Genera el listado de usuarios bloqueados por el usuario

let usuariosBloqueados: Set(UsuarioRegistrado) = self.bloqueado

Incluir/Excluir usuario de bloqueados

Bloquea o desbloquea a un usuario siempre que no sea él mismo, incluyéndolo o excluyéndolo de su listado de usuarios bloqueado. Además en caso de bloquear al usuario y que este estuviese destacado por él, deja de estar destacado. Una vez desbloqueado podría volver a realizar acciones que bloqueado no podía. Estas acciones incluyen enviar correos al usuario que ya no le bloquea y enviar solicitudes de entrada, siempre y cuando se cumplan otras condiciones, en los proyectos creados por el usuario que ya no le bloquea.



Operación: incluirExcluirUsuarioDeBloqueados(usuario: UsuarioRegistrado, bloquear: Boolean)
Descripción: Incluye o excluye a un usuario de su propia lista de usuarios bloqueados
Precondiciones:

El usuario a incluir no es él mismo y no está ya incluido

```
if bloquear then
    usuario <> self and self.bloqueado -> excludes(usuario)
endif
```

El usuario a excluir no es él mismo y ya está incluido.

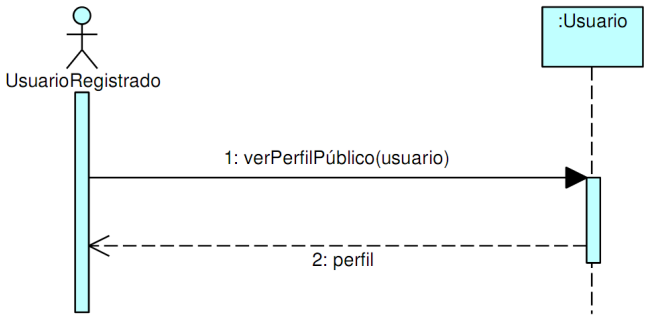
```
if not bloquear then
    usuario <> self and self.bloqueado -> includes(usuario)
endif
```

Postcondiciones:
Incluye al usuario en la lista de bloqueados propios si no lo estaba o lo excluye si ya lo estaba.
En caso de incluirlo si estaba destacado deja de estarlo.

```
if bloquear then
    self.bloqueado -> includes(usuario)
    self.destacado -> excludes(usuario)
else
    self.bloqueado -> excludes(usuario)
endif
```

Ver perfil usuario

Muestra los datos introducidos y indicados como visibles para el resto de usuarios del propietario del perfil. Solo los usuarios registrados tiene acceso. Aunque los usuarios bloqueados por el propietario del perfil tampoco tiene permiso para visualizar los datos del perfil si así lo ha indicado el propietario.



Operación: verPerfilPúblico(usuario: UsuarioRegistrado) : TupleType(nombreUsuario: String, fechaNacimiento: Fecha, nombreReal: String, apellidos: String, dirección: String, emailAlternativo: String, imagenUsuario: Imagen, enlaces: Set(Enlace), experiencia: Set(Experiencia), educación: Set(Educación), méritos: Set(Mérito), referencias: Set(Referencia)), referenciasPropia: Referencia, proyectosPropios: Set(Proyecto), proyectosComoMiembro: Set(Proyecto))
Descripción: Muestra toda la información pública del usuario
Precondiciones:

El usuario existe en el sistema

```
UsuarioRegistrado.allInstances -> exists(u | u = usuario)
```

El usuario visitante del perfil no está bloqueado por el propietario del perfil.

```
usuario.bloqueado -> excludes(self)
```

Salida:

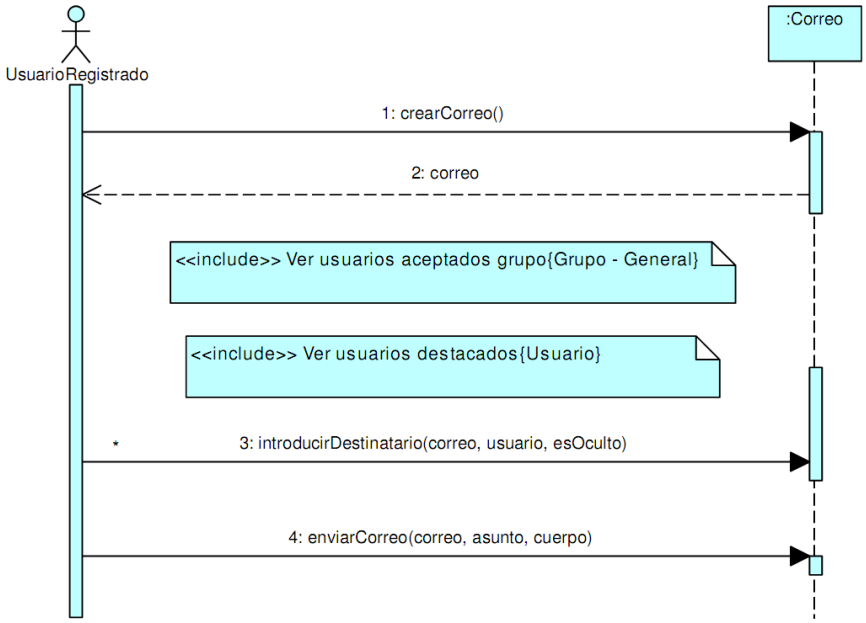
Muestra toda la información pública del usuario dependiendo de la visibilidad de los datos que haya indicado el propietario del perfil

```
let perfil : TupleType(nombreUsuario: String, fechaNacimiento: Fecha, nombreReal: String, apellidos: String,
                       dirección: String, emailAlternativo: String, imagenUsuario: Imagen, enlaces: Set(Enlace),
                       experiencia: Set(Experiencia), educación: Set(Educación), méritos: Set(Mérito),
                       referencias: Set(Referencia), referenciaPropia: Referencia,
                       proyectosPropios: Set(Proyecto), proyectosComoMiembro: Set(Proyecto)) in

perfil.nombreUsuario = usuario.nombreUsuario
if usuario.visibilidadFechaNac then
   perfil.fechaNacimiento = usuario.fechaNacimiento
else
   perfil.fechaNacimiento -> oclIsUndefined()
endif
if usuario.visibilidadEmail then
   perfil.emailAlternativo = usuario.emailAlternativo
else
   perfil.emailAlternativo -> oclIsUndefined()
endif
if usuario.visibilidadDatosPersonales then
   perfil.nombreReal = usuario.nombreReal
   perfil.apellidos = usuario.apellidos
   perfil.dirección = usuario.dirección
else
   perfil.nombreReal -> oclIsUndefined()
   perfil.apellidos -> oclIsUndefined()
   perfil.dirección -> oclIsUndefined()
endif
if usuario.visibilidadProyectos then
   perfil.proyectosPropios = Proyecto.allInstances -> select(p | p.creador.usuarioRegistrado = usuario)
   perfil.proyectosComoMiembro = Proyecto.allInstances -> select(p | p.grupo.miembroAceptado.usuarioRegistrado
                                                                    -> includes(usuario))
else
   perfil.proyectosPropios -> oclIsUndefined()
   perfil.proyectosComoMiembro -> oclIsUndefined()
endif

perfil.imagenUsuario = usuario.imagenUsuario and perfil.enlaces = usuario.enlace and
perfil.experiencia = usuario.experiencia and
perfil.educación = usuario.educación and perfil.meritos = usuario.mérito and
perfil.referencias = Referencia.allInstances -> select(r | r.referenciado = usuario and
                                                           r.estado = EstadoReferencia::Aprobada) and
perfil.referenciaPropia = Referencia.allInstances -> select(r | r.referenciado = usuario and r.referenciador = self)
```

3.4.2. Diagramas de secuencia de correo

Enviar correo

Crea un correo que puede ser enviado a uno o más usuarios registrados. El correo se compone de un asunto y un cuerpo. Todo destinatario puede ser introducido manualmente y/o elegido de un listado de contactos. Estos contactos están clasificados por proyectos. Estos proyectos a su vez están clasificados según la membresía del emisor dentro de estos, siendo administrador, miembro, invitado o candidato o también proyectos que haya destacado. Una vez escogido un proyecto le serán mostrados los usuarios que participan en ese proyecto clasificados por la membresía en el proyecto. Los usuarios que vea dependerá de la membresía que tenga el propio usuario dentro del proyecto escogido. Como administrador podrá ver a todos los usuarios del proyecto, como miembro e invitado podrá verlos a todos menos los candidatos y como candidato o no participante solo verá al administrador. Además podrá escoger entre los contactos que tenga destacados. Por último, tanto los contactos introducidos manualmente como los escogidos pueden ser añadidos como visibles u ocultos. Como visibles lo serán para el resto de destinatarios del correo, en cambio como ocultos no serán mostrados al resto de destinatarios.



Operación: crearCorreo()
Descripción: Crea un objeto correo para rellenar los campos
Precondiciones: -
Salida:
Crea un objeto correo vacío

let correo : Correo -> oclIsUndefined() in
correo.oclIsNew() and correo.oclIsTypeOf(Correo) and
correo.identificador = Correo.allInstances().identificador -> max() + 1

Operación: introducirDestinatario(correo: Correo, usuario: UsuarioRegistrado, esOculto: Boolean)
Descripción: El destinatario es agregado al correo
Precondiciones:
El correo debe existir en el sistema
Correo.allInstances() -> exists(c | c = correo)

El usuario debe existir en el sistema
UsuarioRegistrado.allInstances() -> exists(u | u = usuario)

El destinatario no puede tener bloqueado al emisor
usuario.bloqueado -> excludes(usuario)

El destinatario no se ha introducido ya en el correo
correo.infoCorreo.destinatario -> excludes(usuario)

Postcondiciones:
El destinatario es agregado al correo indicando si este será como copia oculta o será visible para el resto de destinatarios.

let infoCorreo : infoCorreo -> oclIsUndefined() in
infoCorreo.oclIsNew() and infoCorreo.oclIsTypeOf(InfoCorreo) and
infoCorreo.leido = false and infoCorreo.destacado = false and infoCorreo.oculto = esOculto and
infoCorreo.correo = correo and infoCorreo.emisor = self and infoCorreo.destinatario = usuario

Operación: enviarCorreo(correo: Correo, asunto: String, cuerpo: String)
Descripción: El correo es enviado a los destinatarios
Precondiciones:

El correo debe existir en el sistema
Correo.allInstances() -> exists(c | c = correo)

El asunto y cuerpo no están vacíos
not(asunto -> oclIsUndefined() and cuerpo -> oclIsUndefined())

El asunto tiene una longitud de entre 1 y 512 caracteres
asunto -> size() >= 1 and asunto -> size() <= 512

El cuerpo tiene una longitud de entre 1 y 307200 caracteres (300KB)
cuerpo -> size() >= 1 and cuerpo -> size() <= 307200

Postcondiciones:

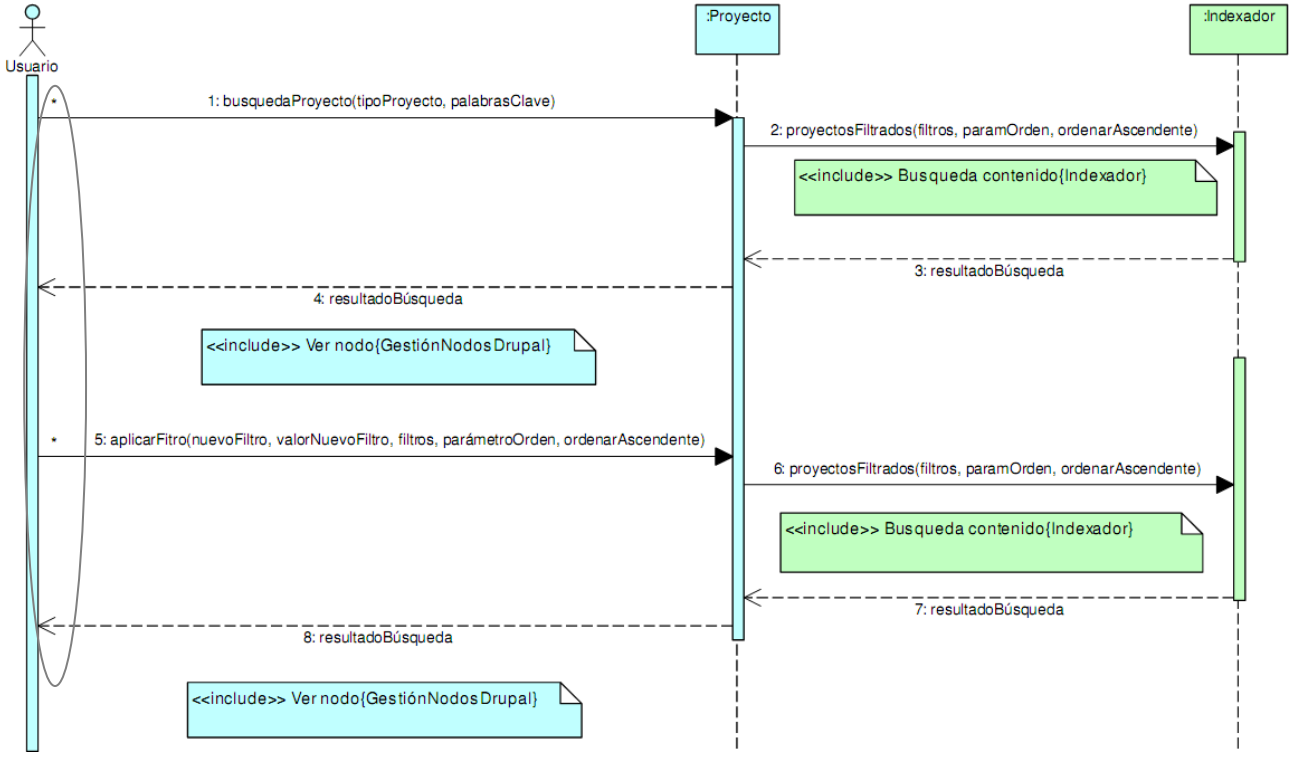
El correo es rellenado con los datos necesarios y enviado a los destinatarios

correo.asunto = asunto and correo.fecha = fechaActual() and correo.cuerpo = cuerpo and
correo.leidoEmisor = false and correo.infoCorreo.emisor.correoEnviado -> includes(correo) and

3.4.3. Diagramas de secuencia de proyecto

Búsqueda proyectos

Todo el contenido de los proyectos ha sido previamente indexado por el buscador. El usuario introduce una o más palabras clave para empezar la búsqueda y puede indicar también un tipo de proyecto para acotar aun más la búsqueda. Las palabras clave indexadas vienen del contenido de la descripción breve y completa, candidatos y título. El sistema además de devolver los proyectos que cumplen los criterios indicados, también muestra los filtros que se pueden aplicar para acotar aun más la búsqueda. Estos filtros se adaptan a los resultados y son llamados Facets.



Operación: busquedaProyecto(tipoProyecto: TipoProyecto, palabrasClave: Set(String)) :
Set(TupleType(proyectos: Set(Proyecto), filtrosAplicados: Set(TupleType(filtro: String, valor: oclAny)),
filtrosDisponibles: Set(TupleType(filtro: String, valor: oclAny)),
orden: TupleType(parámetro: String, ordenarAscendente: Boolean)))

Descripción: Realiza una búsqueda de proyectos a través de las palabras indexadas en el título, descripción breve, descripción completa y candidatos del proyecto

Precondiciones: -

Salida:

Muestra el listado de los proyectos que cumplen los criterios indicados además de los filtros aplicados con tal de adaptar los nuevos filtros disponibles a mostrar

```
let filtros : Set(TupleType(filtro: String, valor: oclAny)) in
filtros -> includes(Tuple{filtro = 'Palabras clave', valor = palabrasClave})
if not tipoProyecto -> oclIsUndefined() then
    filtros -> includes(Tuple{filtro = 'Tipo de proyecto', valor = tipoProyecto})
endif
let  resultadoBúsqueda : Set(TupleType(proyectos: Set(Proyecto),
                                       filtrosAplicados: Set(TupleType(filtro: String, valor: oclAny()))),
                                filtrosDisponibles: Set(TupleType(filtro: String, valor: oclAny())),
                                orden: TupleType(parámetro: String, ordenarAscendente: Boolean))) =
Indexador::proyectosFiltrados(filtros, oclIsUndefined(), oclIsUndefined())
```

Operación: aplicarFiltro(nuevoFiltro: String, valorNuevoFiltro: oclAny,
filtros: Set(TupleType(filtro: String, valor: oclAny)),
paramOrden: String, ordenarAscendente: Boolean) :
Set(TupleType(proyectos: Set(Proyecto), filtrosAplicados: Set(TupleType(filtro: String, valor: oclAny)),
filtrosDisponibles: Set(TupleType(filtro: String, valor: oclAny)),
orden: TupleType(parámetro: String, ordenarAscendente: Boolean)))

Descripción: Realiza una búsqueda de proyectos aplicando los filtros anteriores además de uno nuevo y ordenando el resultado.

Precondiciones: -

Salida:

Muestra el listado de los proyectos que cumplen los criterios indicados además de los filtros aplicados con tal de adaptar los nuevos filtros disponibles a mostrar. A los criterios anteriormente utilizados se le aplica uno nuevo y el resultado es ordenado según el parámetro indicando en orden ascendente o descendente.

```
let filtros : Set(TupleType(filtro: String, valor: oclAny)) = filtros in
filtros -> includes(Tuple{filtro = nuevoFiltro, valor = valorNuevoFiltro})

let  resultadoBúsqueda : Set(TupleType(proyectos: Set(Proyecto),
                                       filtrosAplicados: Set(TupleType(filtro: String, valor: oclAny)),
                                       filtrosDisponibles: Set(TupleType(filtro: String, valor: oclAny)),
                                       orden: TupleType(parámetro: String, ordenarAscendente: Boolean))) =
                                Indexador::proyectosFiltrados(filtros, paramOrden, ordenarAscendente)
```

Operación: proyectosFiltrados(filtros: Set(TupleType(filtro: String, valor: oclAny)),
paramOrden: String,
ordenarAscendente: Boolean) :
Set(TupleType(proyectos: Set(Proyecto), filtrosAplicados: Set(TupleType(filtro: String, valor: oclAny)),
filtrosDisponibles: Set(TupleType(filtro: String, valor: oclAny)),
orden: TupleType(parámetro: String, ordenarAscendente: Boolean)))

Descripción: Realiza una búsqueda de proyectos aplicando los filtros y ordenando el resultado.

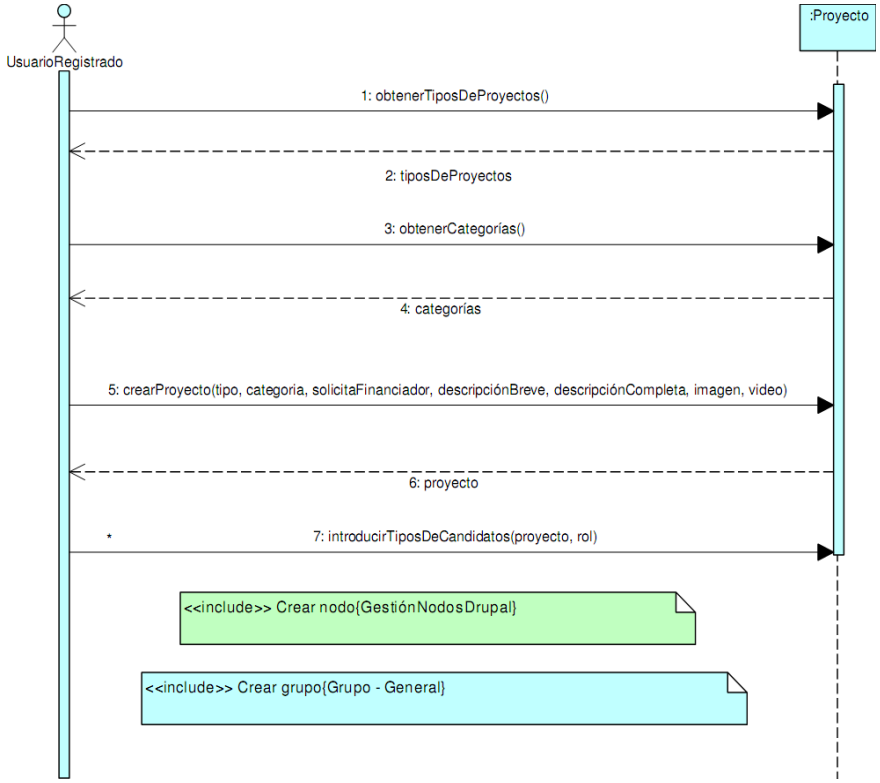
Precondiciones: -

Salida:

Retorna el resultado de aplicar filtros a la información de proyectos indexada en el indexador escogido. Además ordena el resultado por el parámetro indicado paramOrden y de forma ascendente o descendente según el parámetro ordenarAscendente. El resultado también lleva un listado de filtros que se pueden aplicar a los proyectos resultantes de la búsqueda.

Crear proyecto

Crea un proyecto introduciendo el tipo, ya sea social(sin ánimo de lucro) o emprendedor(beneficio económico). Además puede clasificarlo por categorías predefinidas. El proyecto también se compone de una descripción breve y otra completada, además de una imagen y el enlace a un vídeo relacionado con el proyecto. Por último introduce también los candidatos que serán los roles o actividades necesarias para hacer progresar el proyecto. Existe un tipo de candidato especial que es el financiador, que además de financiar el proyecto tiene la posibilidad de participar en él.



Operación: obtenerTiposDeProyecto(): Set(TipoProyecto)
Descripción: Obtiene los tipos de proyecto que existen
Precondiciones: -
Salida:
let tiposDeProyecto : Set(TipoProyecto) = TipoProyecto.allInstances()

Operación: obtenerCategorías(): Set(Categoría)
Descripción: Obtiene las categorías que existen
Precondiciones: -
Salida:
let tiposDeProyecto : Set(Categoría) = Categoria.allInstances()

Operación: crearProyecto(tipo: TipoProyecto, categoría: Categoria, solicitaFinanciador: Boolean, descripciónBreve: String, descripciónCompleta: String, imagen: Imagen, video: Video): Proyecto
Descripción: El usuario crea un proyecto que es publicado de forma que sea accesible por el resto de usuarios.
Precondiciones:
El tipo de proyecto, categoría y solicitaFinanciador no pueden estar vacíos
not tipo -> oclIsUndefined() and not categoría -> oclIsUndefined() and not solicitaFinanciador -> oclIsUndefined()

La descripción breve y completa no pueden estar vacías. La descripción breve debe tener una longitud máxima de 300 caracteres y la completa de 150000 caracteres
not descripciónBreve -> oclIsUndefined() and not descripciónCompleta -> oclIsUndefined() and descripciónBreve -> size() >= 1 and descripciónBreve -> size() <= 300 and descripciónCompleta -> size() >= 1 and descripciónCompleta -> size() <= 150000

Postcondiciones:
El proyecto es agregado en el sistema, haciendo al creador administrador del proyecto.
let proyecto : Proyecto -> oclIsUndefined() in
proyecto.oclIsNew() and proyecto.oclIsTypeOf(Proyecto) and proyecto.creador = self and proyecto.tipoProyecto = tipo and proyecto.descripciónBreve = descripciónBreve and proyecto.descripciónCompleta = descripciónCompleta and proyecto.imagen = imagen and proyecto.video = video and proyecto.categoría = categoría and proyecto.solicitaFinanciador = solicitaFinanciador and proyecto.identificador = Proyecto.allInstances.identificador -> max() + 1

Operación: introducirTiposDeCandidatos(proyecto: Proyecto, rol: String)
Descripción: Añade un tipo de rol o actividad al proyecto
Precondiciones:
El proyecto existe en el sistema
Proyecto.allInstances() -> exists(p | p = proyecto)

El usuario es el administrador del proyecto
proyecto.creador = self

No existe ya el rol introducido en el proyecto
proyecto.roles -> excludes(rol)

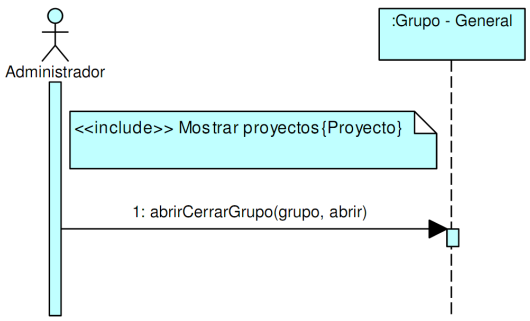
El rol no puede estar vacío y debe tener una longitud de entre 1 y 60 caracteres
not rol -> oclIsUndefined() and rol -> size() >= 1 and rol -> size() <= 60

Postcondiciones:
El rol es añadido al proyecto
proyecto.roles -> includes(rol)

3.4.4. Diagramas de secuencia general de grupo

Abrir/Cerrar grupo

El grupo puede ser abierto o cerrado a nuevas solicitudes



Operación: abrirCerrarGrupo(grupo: Grupo, abrir: Boolean)
Descripción: Abre un grupo que previamente había estado cerrado
a nuevas solicitudes o lo cierra si es que estaba abierto.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El administrador del grupo es el propio usuario
grupo.creador.usuarioRegistrado = self

El proyecto está abierto o cerrado
grupo.estado = EstadoGrupo::Abierto or
grupo.estado = EstadoGrupo::Cerrado

Postcondiciones:
El grupo queda abierto o cerrado a nuevas solicitudes dependiendo de su estado anterior.
if abrir then
grupo.estado = EstadoGrupo::Abierto
else
grupo.estado = EstadoGrupo::Cerrado
endif

Crear grupo

Un único grupo es creado para el proyecto recién creado y este se encarga de gestionar las vías de comunicación como chat, muro además de los eventos. Además gestiona la asociación de usuarios con el grupo a través de las distintas membresías y por último la recaudación entre otros aspectos.

Operación: crearGrupo(proyecto: Proyecto, requiereDescripciónAptitudes: Boolean)
Descripción: Crea el grupo asociado a un proyecto
Precondiciones:
Debe existir el proyecto en el sistema
Proyecto.allInstances() -> exists(p | p = proyecto)

El administrador es el creador del proyecto
proyecto.creador = self

El proyecto no tiene ya asociada una instancia de grupo
proyecto.grupo -> oclIsUndefined()

Postcondiciones:
El grupo es creado y asociado al proyecto

g.oclIsNew() and g.oclIsTypeOf(Grupo) and g.identificador = proyecto.identificador and g.totalMiembros = 0 and g.totalInvitados = 0 and g.totalCandidatos = 0 and g.requiereAptitudesCandidato = requiereDescripciónAptitudes and g.estado = EstadoGrupo::Abierto and g.recaudadoTotal = 0 and g.totalFinanciadores = 0 and g.objetivoRecaudación = 0 and g.fechaObjetivo -> oclIsUndefined() and g.cuenta = '' and g.divisa = '' and g.recaudaciónActiva = false and g.sistemaRecompensasActivo = false and g.inversiónMínima = 0 and g.destacados = 0 and g.porcentajeObjetivosCumplidos = 0 and g.visibildiadObjetivos = 0

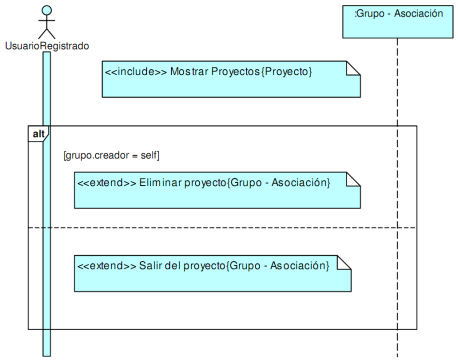
let infoUsuario : InfoUsuario = g.infoUsuario -> select(i | i.usuarioRegistrado = self) in
infoUsuario.membresía = TipoUsuario::Administrador and infoUsuario.pagoPendiente = 0 and infoUsuario.inversión = 0 and infoUsuario.devoluciónPendiente = 0 and infoUsuario.destacadaProyecto = false and infoUsuario.rol = '' and infoUsuario.fecha = fechaActual() and infoUsuario.estado = EstadoUsuario::Aceptado and infoUsuario.oclIsTypeOf(Administrador) and g.creador = infoUsuario

```
if proyecto.roles -> size() > 20
   g.maxMiembros  =  proyecto.roles -> size()
else
   g.maxMiembros  = 20
endif
if proyecto.roles -> size() > 30
   g.maxInvitados =  proyecto.roles -> size()
else
   g.maxInvitados = 30
endif
if proyecto.roles -> size() > 50
   g.maxCandidatos  =  proyecto.roles -> size()
else
   g.maxCandidatos = 50
endif
```

let chat : Chat -> oclIsUndefined() in
let muro : MuroMensajes -> oclIsUndefined() in
chat.oclIsNew() and chat.oclIsTypeOf(Chat) and chat.visibilidad = 'VisibleParaAdministradorMiembrosInvitados' and g.chat = chat and muro.oclIsNew() and muro.oclIsTypeOf(MuroMensajes) and g.muroMensajes = muro and proyecto.grupo = g

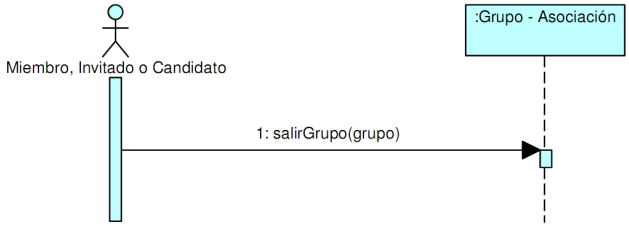
3.4.5. Diagramas de secuencia asociación a grupo

Salir de/Eliminar grupo



Salir del proyecto

Todo miembro, invitado o candidato de un proyecto podrá salir en cualquier momento. A partir de entonces dejará de participar en este y tendrá el mismo acceso que cualquier otro usuario registrado que no participe en el proyecto. Por tanto, para volver a participar debe volver a enviar una solicitud.



Operación: salirGrupo(grupo: Grupo)
Descripción: El usuario sale del grupo dejando de tener acceso a las vías de comunicación de este, como cualquier otro usuario registrado.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El usuario es candidato, invitado o miembro del grupo

```
grupo.candidatoAceptado.usuarioRegistrado -> includes(self) or
grupo.invitadoAceptado.usuarioRegistrado -> includes(self) or
grupo.miembroAceptado.usuarioRegistrado -> includes(self)
```

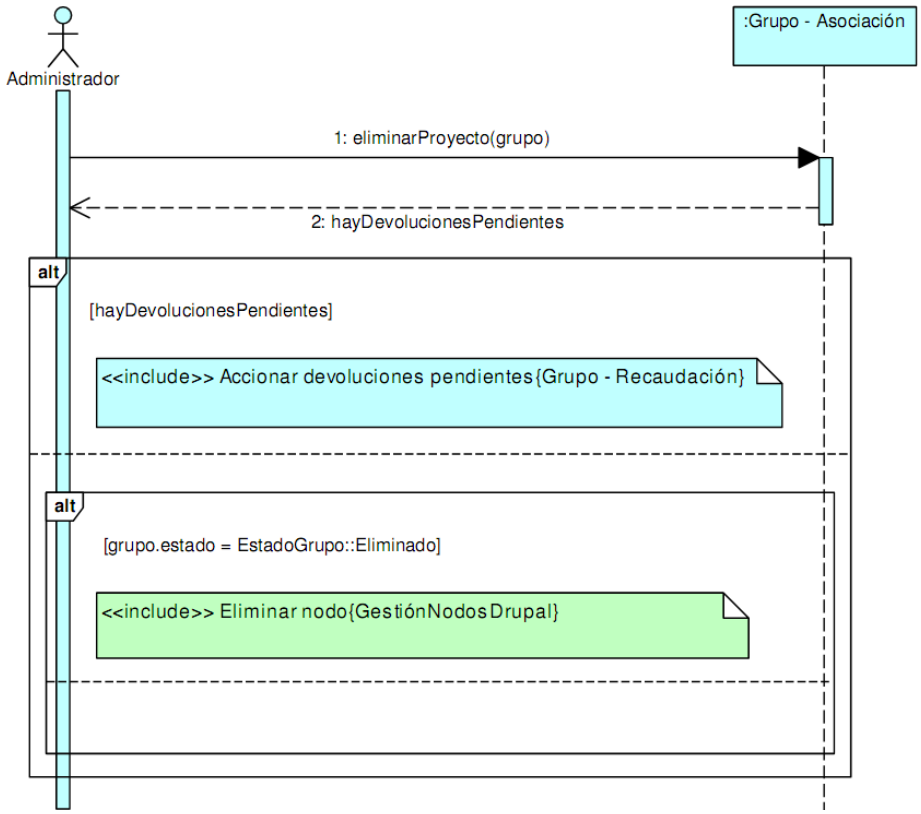
Postcondiciones:

En caso de ser candidato, invitado o miembro sale del grupo y no tiene acceso a este ni ningún elemento de este.

```
let infoUsuario : InfoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
infoUsuario.membresia = TipoUsuario::NoParticipa  and infoUsuario.rol = '' and infoUsuario.fecha -> oclIsUndefined()

if grupo.miembroAceptado -> includes(infoUsuario) then
   grupo.miembroAceptado -> excludes(infoUsuario) and grupo.evento.miembroParticipa -> excludes(infoUsuario) and
   grupo.conexiónChat.infoUsuario -> excludes(infoUsuario) and grupo.muroMensajes.infoUsuario
   -> excludes(infoUsuario) and grupo.totalMiembros = grupo.totalMiembros - 1
elseif grupo.invitadoAceptado -> includes(infoUsuario) then
   grupo.conexiónChat.infoUsuario -> excludes(infoUsuario) and grupo.solicitudAceptada.invitado
   -> excludes(infoUsuario) and grupo.invitadoAceptado -> excludes(infoUsuario)

   let solicitudAceptada : SolicitudAceptada = grupo.solicitudAceptada -> select(s | s.invitado = infoUsuario) in
   if not solicitudAceptada -> oclIsUndefined() then
      solicitudAceptada.invitado -> oclIsUndefined() and solicitudAceptada.grupo -> oclIsUndefined() and
       SolicitudAceptada.allInstances() -> excludes(solicitudAceptada)
   endif
   grupo.totalInvitados = grupo.totalInvitados - 1
elseif grupo.candidatoAceptado -> includes(infoUsuario) then
   grupo.candidatoAceptado -> excludes(infoUsuario)

   let solicitud : Solicitud = grupo.solicitudPendiente -> select(s | s.candidato = infoUsuario) in
   solicitud.candidato -> oclIsUndefined() and solicitud.grupo -> oclIsUndefined() and Solicitud.allInstances()
    -> excludes(solicitud) and  grupo.totalCandidatos = grupo.totalCandidatos - 1
endif

infoUsuario.oclIsTypeOf(InfoUsuario)
```

Eliminar proyecto

Elimina el proyecto en caso darse las condiciones adecuadas sino queda pendiente de eliminación. Si hubiera recompensas solicitadas, pago completados de miembros de hace menos de 60 días o devoluciones pendientes, el proyecto quedaría estado Pendiente de eliminación hasta que se resolvieran estos supuestos.



Operación: eliminarProyecto(grupo: Grupo)
Descripción: Elimina el proyecto del sistema.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El administrador lo es del grupo

```
grupo.creador.usuarioRegistrado = self
```

Postcondiciones:

Elimina el proyecto en caso darse las condiciones adecuadas sino queda pendiente de eliminación.
Si hubiera recompensas solicitadas, pago completados de miembros de hace menos de 60 días o devoluciones pendientes, el proyecto quedaría estado Pendiente de eliminación hasta que se resolvieran estos supuestos.

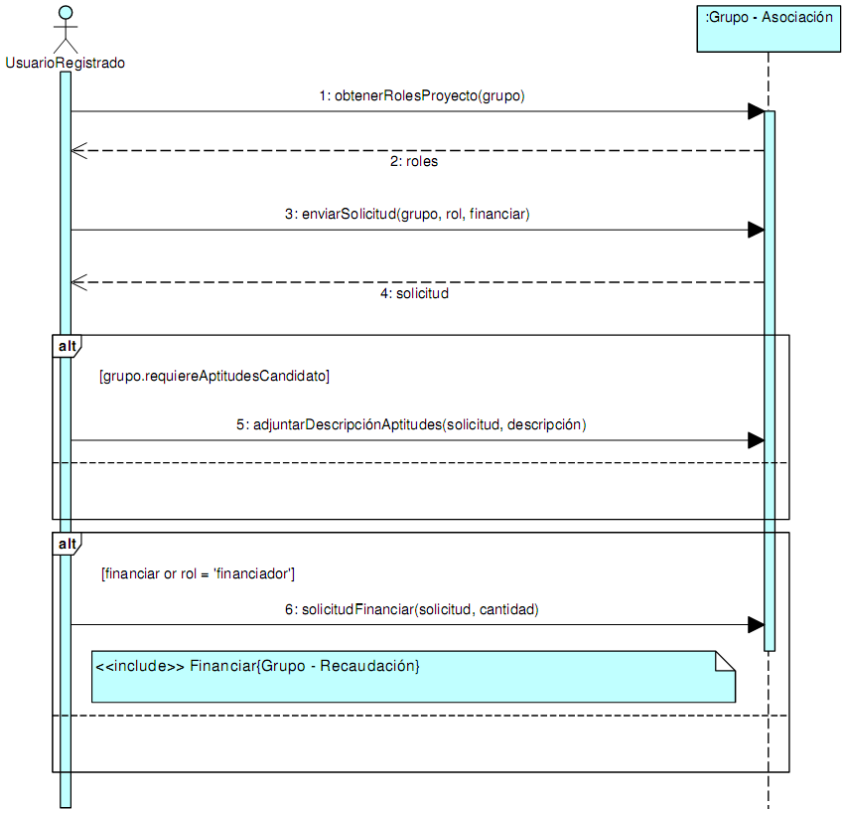
```
Solicitud.allInstances() -> excludesAll(grupo.solicitud -> union(grupo.solicitudRechazada)
 -> union(grupo.solicitudAprobada)) and
grupo.solicitudPendiente.candidato -> oclIsUndefined and grupo.solicitudPendiente -> isEmpty() and
grupo.solicitudRechazada.candidato -> oclIsUndefined and grupo.solicitudRechazada -> isEmpty() and
grupo.solicitudAceptada.invitado -> oclIsUndefined and grupo.solicitudAceptada -> isEmpty() and
grupo.invitadoAceptado -> forAll(i | i.membresía = TipoUsuario::NoParticipa and i.rol = '' and i.oclIsTypeOf(InfoUsuario)) and
grupo.candidatoAceptado -> forAll(i | i.membresía = TipoUsuario::NoParticipa and i.rol = '' and i.oclIsTypeOf(InfoUsuario)) and
grupo.invitadoAceptado -> isEmpty() and grupo.invitadoRechazado -> isEmpty() and
grupo.candidatoAceptado -> isEmpty() and grupo.candidatoRechazado -> isEmpty()

 if grupo.recompensa.recompensaSolicitadas -> isEmpty() and
  grupo.pagoCompletado -> select(p | fechaActual() - p.fecha < 60) -> isEmpty and grupo.devolución -> isEmpty() then
    grupo.miembro -> isEmpty() and grupo.miembroRechazado -> isEmpty() and
    grupo.creador -> excludes(grupo.creador) and Evento.allInstances() -> excludesAll(grupo.evento) and
    MensajeChat.allInstances() -> excludesAll(grupo.chat.MensajeChat) and
    Chat.allInstances() -> excludesAll(grupo.chat) and
    ComentarioMuro.allInstances() -> excludesAll(grupo.muroMensajes.comentarioMuro) and
    MensajeMuro.allInstances() -> excludesAll(grupo.muroMensajes.mensajeMuro) and
    MuroMensajes.allInstances() -> excludesAll(grupo.muroMensajes) and
    Proyecto.AllInstances -> excludes(grupo.proyecto) and
    Transacciones.allInstances() -> excludesAll(grupo.Transacciones) and
    Recompensas.allInstances() -> excludesAll(grupo.recompensas) and
    Objetivo.allInstances() -> excludes(grupo.objetivo) and
    InfoUsuario.allInstances() -> excludes(grupo.infoUsuario) and Grupo.allInstances() -> excludes(grupo) and
    grupo.estado = EstadoGrupo::Eliminado
 else
    grupo.estado = EstadoGrupo::PendienteEliminación
    let miembrosExpulsión : Set(Miembro) = grupo.miembroAceptado
                          -> reject(m | m.pagoCompletado -> exists(p | p.grupo = grupo and fechaActual() - p.fecha <= 60)) in
    grupo.miembroAceptado -> excludesAll(miembrosExpulsión) and
    grupo.miembroAceptado -> forAll(m | m.estado = EstadoUsuario::PendienteExpulsión)
    let miembrosDevoluciones : Set(Miembro) = grupo.miembroAceptado
                          -> select(m | m.pagoCompletado -> exists(p | p.grupo = grupo and fechaActual() - p.fecha <= 60)) in
    miembrosDevoluciones
      -> iterate(m | m.usuarioRegistrado.pagoCompletado -> select( p | p.grupo = grupo and fechaActual() - p.fecha <= 60)
                  -> forAll(p | p.oclIsTypeOf(Devolución) and m.inversión = m.inversión - p.cantidad and
                                m.devoluciónPendiente = m.devoluciónPendiente + p.cantidad and p.estado = 'Sin realizar'
                                  and p.recompensaSolicitada.estado = EstadoRecompensa::PendienteDeDevoluciónDePago
                      )
            )
 endif
```

Salida:

```
 if grupo.devolución -> notEmpty() then
   hayDevolucionesPendientes = true
 else
   hayDevolucionesPendientes = false
 endif
```

Enviar solicitud

Envía una solicitud de participación en el proyecto siempre y cuando se cumplan unos requisitos. Estos requisitos son que no participe ya como administrador, miembro o invitado o esté expulsado del proyecto. Tampoco podrá enviar la solicitud si el administrador del proyecto ha bloqueado al usuario. Además, el número de candidatos no puede superar al máximo establecido. Por último, el grupo del proyecto debe tener el estado abierto a nuevas solicitudes. Una vez cumplidos estos requisitos y enviada la solicitud, el usuario empieza a participar como candidato. Como candidato tiene que esperar a que el administrador del proyecto acepte o rechace la solicitud. Si es aceptada el candidato es promovido a invitado, en cambio, si es rechazada, el candidato estará expulsado por un periodo máximo de 4 meses. Hasta pasado ese tiempo o que el administrador le desbloquee del proyecto, es decir le quite la expulsión, no podrá volver a enviar una solicitud.



Operación: obtenerRoles(grupo : Grupo): Set(String)
Descripción: Obtiene los roles necesitados por el proyecto
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El grupo está abierto a nuevas solicitudes
grupo.estado = EstadoGrupo::Abierto

Salida:
Retorna una colección de roles
let roles : Set(String) = grupo.proyecto.roles
if grupo.proyecto.solicitaFinanciador then
roles = roles -> union('financiador')
endif

Operación: enviarSolicitud(grupo : Grupo, rol : String, financiar : Boolean)
Descripción: Un usuario registrado solicita el ingreso en el grupo.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El grupo está abierto a nuevas solicitudes
grupo.estado = EstadoGrupo::Abierto

El usuario registrado no es ya administrador, miembro, invitado o candidato del grupo.
grupo.administrador.usuarioRegistrado -> excludes(self) and
grupo.miembroAceptado.usuarioRegistrado -> excludes(self) and
grupo.invitadoAceptado.usuarioRegistrado -> excludes(self) and
grupo.candidatoAceptado.usuarioRegistrado -> excludes(self)

El usuario registrado no está expulsado del grupo.
grupo.miembroRechazado.usuarioRegistrado -> excludes(self) and
grupo.invitadoRechazado.usuarioRegistrado -> excludes(self) and
grupo.candidatoRechazado.usuarioRegistrado -> excludes(self)

El total de candidatos es menor que el máximo de candidatos del grupo
grupo.totalCandidatos < grupo.maxCandidatos

El administrador del grupo no tiene bloqueado al usuario
grupo.creador.usuarioRegistrado.bloqueado -> excludes(self)

Postcondiciones:
El usuario se convierte en candidato del grupo.
let infoUsuario : InfoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
infoUsuario.membresía = TipoUsuario::Candidato and infoUsuario.estado = EstadoUsuario::Aceptado and
infoUsuario.fecha = fechaActual() and infoUsuario.oclIsTypeOf(Candidato)

let solicitud : Solicitud -> oclIsUndefined() in
solicitud.oclIsNew() and solicitud.oclIsTypeOf(Solicitud) and infoUsuario.solicitud -> includes(solicitud) and
solicitud.fechaEnvio = fechaActual() and solicitud.identificador = Solicitud.allInstrances().identificador -> max() + 1 and
grupo.solicitudPendiente -> includes(solicitud) and solicitud.leida = false and solicitud.destacada = false
grupo.totalCandidatos = grupo.totalCandidatos + 1

Operación: adjuntarDescripciónAptitudes(solicitud: Solicitud, descripción : String)
Descripción: Adjunta la descripción de aptitudes de la solicitud
Precondiciones:

Debe existir la solicitud en el sistema
Solicitud.allInstances() -> exists(s | s = solicitud)

La solicitud pertenece al usuario
solicitud.candidato.usuarioRegistrado = self

Postcondiciones:
Adjunta la descripción de aptitudes de la solicitud del usuario
solicitud.descripciónAptitudes = descripción

Operación: solicitudFinanciar(solicitud: Solicitud, cantidad: Integer)
Descripción: Adjunta la cantidad con la que financiará el proyecto
Precondiciones:

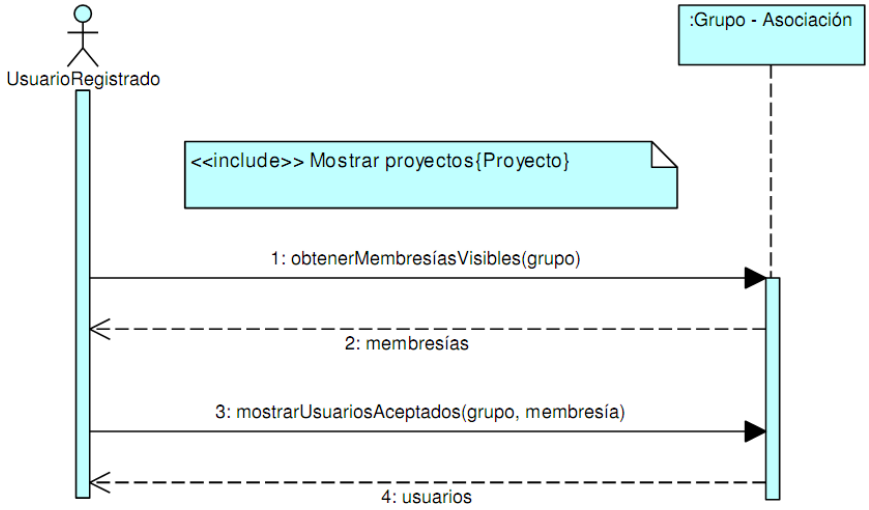
Debe existir la solicitud en el sistema
Solicitud.allInstances() -> exists(s | s = solicitud)

La solicitud pertenece al usuario
solicitud.candidato.usuarioRegistrado = self

Postcondiciones:
Adjunta la cantidad con la que financiará el proyecto
solicitud.inversión = cantidad

Ver usuarios aceptados de grupo

Muestra los usuarios aceptados del grupo, dependiendo de la membresía del propio usuario en el grupo. El administrador del proyecto podrá ver todos los usuarios que participen en el proyecto en cambio miembros e invitados los verán todos menos los candidatos. Y el resto solo verán el administrador. La información que es mostrada es además del propio usuario, es la fecha de ingreso y el rol o actividad que desempeña. El administrador tiene acceso a información complementaria que está formada por el pago pendiente, pagos completados y devoluciones de cada usuario participante hacia el proyecto.



Operación: obtenerMembresíasVisibles(grupo: Grupo) : Set(TipoUsuario)
Descripción: Muestra el listado de membresías de las que el usuario puede ver el listado.
Precondiciones: -
Salida:

```
let infoUsuario : infoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
let membresías : Set(TipoUsuario) -> oclIsUndefined() in
if grupo.creador = infoUsuario then
   membresias -> includes(TipoUsuario::Administrador) -> includes (TipoUsuario::Miembro)
   -> includes(TipoUsuario::Invitado) -> includes(TipoUsuario::Candidato)
elseif grupo.miembroAceptado -> includes(infoUsuario) or grupo.invitadoAceptado -> includes(infoUsuario) then
      membresias -> includes(TipoUsuario::Administrador) -> includes (TipoUsuario::Miembro) -> includes(TipoUsuario::Invitado)
  else
      membresias -> includes(TipoUsuario::Administrador)
  endif
```

Operación: mostrarUsuariosAceptados(grupo: Grupo, membresía: TipoUsuario):
Set(TupleType(usuario: UsuarioRegistrado, fechaIngreso: Fecha, rol: String,
pagoPendiente: PagoPendiente,
pagosCompletados: Set(PagoCompletado), devolucionesPendientes: Set(Devolución))
Descripción: Muestra el listado de los usuarios aceptados del grupo
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El administrador del proyecto podrá ver todos los usuarios que participen en el proyecto en cambio miembros e invitados los verán todos menos los candidatos. Y el resto solo verán el administrador.

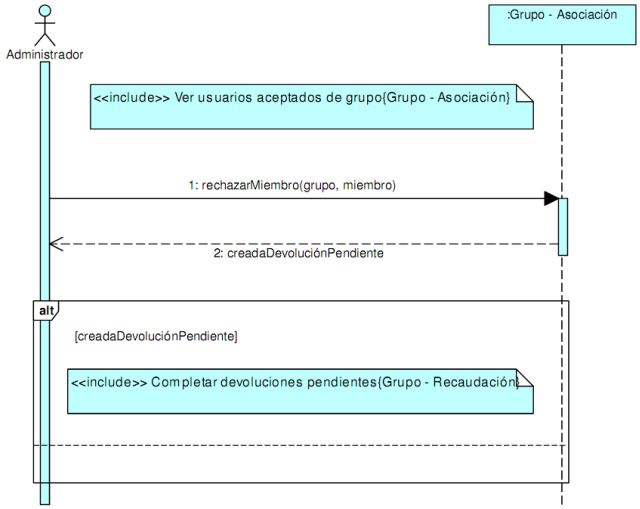
```
let infoUsuario : infoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
if grupo.creador = infoUsuario then
    membresia = TipoUsuario::Administrador or TipoUsuario::Miembro or TipoUsuario::Invitado or TipoUsuario::Candidato
elseif grupo.miembroAceptado -> includes(infoUsuario) or grupo.invitadoAceptado -> includes(infoUsuario) then
    membresia = TipoUsuario::Administrador or TipoUsuario::Miembro or TipoUsuario::Invitado
else
    membresia = TipoUsuario::Administrador
endif
```

Salida:

```
let usuarios : Set(TupleType(usuario: UsuarioRegistrado, rol: String, fechaIngreso: Fecha, pagoPendiente: PagoPendiente,
                             pagosCompletados: Set(PagoCompletado), devolucionesPendientes: Set(devolución)) in
let usuariosAceptados : Set(InfoUsuario) in
  if membresia = TipoUsuario::Administrador then
    usuariosAceptados = grupo.creador
  elseif membresia = TipoUsuario::Miembro then
    usuariosAceptados = grupo.miembroAceptado
  elseif membresia = TipoUsuario::Invitado then
    usuariosAceptados = grupo.invitadoAceptado
  elseif membresia = TipoUsuario::Candidato then
    usuariosAceptados = grupo.candidatoAceptado
  endif
let infoUsuario : infoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in

if grupo.creador = infoUsuario then
  usuariosAceptados
    -> iterate(u | usuarios -> includes(Tuple(usuario = u.usuarioRegistrado, rol = u.rol, fechaIngreso = u.fecha,
                                              pagoPendiente = PagoPendiente.allInstances()
                                                  -> select(p | p.grupo = grupo and p.usuarioRegistrado = u.usuarioRegistrado),
                                              pagosCompletados = PagoCompletado.allInstances()
                                                  -> select(p | p.grupo = grupo and p.usuarioRegistrado = u.usuarioRegistrado),
                                              devolucionesPendientes = Devolución.allInstances()
                                                  -> select(p | p.grupo = grupo and p.usuarioRegistrado = u.usuarioRegistrado)))

                                    )
elseif grupo.miembroAceptado -> includes(infoUsuario) or grupo.invitadoAceptado -> includes(infoUsuario) then
  usuariosAceptados -> iterate(u | usuarios -> includes(Tuple(usuario = u.usuarioRegistrado, rol = u.rol,
                                                              fechaIngreso = u.fecha,
                                                    pagoPendiente = -> oclIsUndefined(),
                                                    pagosCompletados = -> oclIsUndefined(),
                                                    devolucionesPendientes -> oclIsUndefined())
                                    )
else
    usuarios = Set(Tuple(usuario = grupo.creador, rol -> oclIsUndefined() , fechaIngreso -> oclIsUndefined(),
                                    pagoPendiente = -> oclIsUndefined(), pagosCompletados = -> oclIsUndefined(),
                                    devolucionesPendientes -> oclIsUndefined())
endif
```

Rechazar miembro

El miembro es rechazado y por tanto, expulsado del proyecto por el administrador. En caso de que haya financiado el proyecto en los últimos 60 días, estos pagos realizados tendrá que ser devueltos. Los 60 días vienen establecidos por el gestor de transacciones para poder realizar devoluciones. La razón por la que son realizadas las devoluciones es que, si el administrador decide expulsar al miembro, este debería tener derecho a que, hasta donde es posible, lo financiado le sea devuelto ya que el administrador no ha cumplido con la premisa de dejar participar al miembro. Mientras no sean completadas todas las devoluciones el miembro seguirá teniendo acceso habitual al proyecto pero tendrá el estado Pendiente de rechazo. En el momento en que ya no tenga devoluciones será expulsado definitivamente y deberá esperar 4 meses como máximo para volver a solicitar la entrada al proyecto.



Operación: rechazarMiembro(grupo: Grupo, miembro: Miembro): Boolean
Descripción: Expulsa un miembro del grupo y este será expulsado del grupo durante un máximo de 4 meses.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

Debe existir el miembro en el sistema y ser un miembro aceptado en el grupo

```
Miembro.allInstances() -> exists(m | m = miembro) and grupo.miembroAceptado -> includes(miembro)
```

El administrador lo es del propio grupo

```
grupo.creador.usuarioRegistrado = self
```

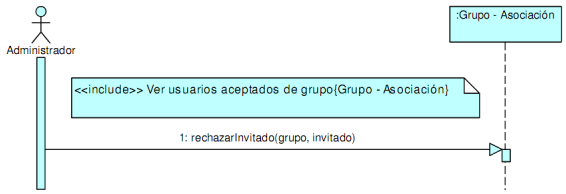
Postcondiciones:

El miembro es expulsado y tendrá el mismo acceso que cualquier otro usuario registrado que no participe en el proyecto.

```
let creadaDevoluciónPendiente : Boolean = false in
let pagosCompletadosRecientes : Set(PagoCompletado) =
  miembro.pagoCompletado -> select(p | fechaActual() - p.fecha <= 60) in
  if pagosCompletadosRecientes -> size() > 0 then
    pagosCompletadoRecientes
      -> iterate(p | infoUsuario.inversión = infoUsuario.inversión - p.cantidad and
                     infoUsuario.devoluciónPendiente = infoUsuario.devoluciónPendiente + p.cantidad and
                     p.oclIsType(Devolución) and p.estado = 'Sin realizar' and
                     p.recompensaSolicitada.estado = EstadoRecompensa::PendienteDeDevoluciónDePago
                )
              creadaDevoluciónPendiente = true and miembro.estado = EstadoUsuario::PendienteRechazo
  else
    grupo.miembroAceptado -> excludes(miembro) and grupo.miembroRechazado -> includes(miembro) and
    grupo.evento.miembroParticipa -> excludes(miembro) and miembro.fecha = fechaActual() and
    grupo.conexiónChat.miembro -> excludes(miembro) and grupo.totalMiembros = grupo.totalMiembros - 1 and
    miembro.estado = EstadoUsuario::Rechazado
  endif
```

Rechazar invitado

El invitado es rechazado y por tanto, expulsado del proyecto por el administrador. Por tanto, deberá esperar 4 meses como máximo para volver a solicitar la entrada al proyecto.



Operación: rechazarInvitado(grupo: Grupo, invitado: Invitado)
Descripción: Expulsa un invitado del grupo y este será expulsado del grupo durante un máximo de 4 meses.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el invitado en el sistema y ser un invitado aceptado en el grupo
Invitado.allInstances() -> exists(i | i = invitado) and grupo.invitadoAceptado -> includes(invitado)

El administrador lo es del propio grupo
grupo.creador.usuarioRegistrado = self

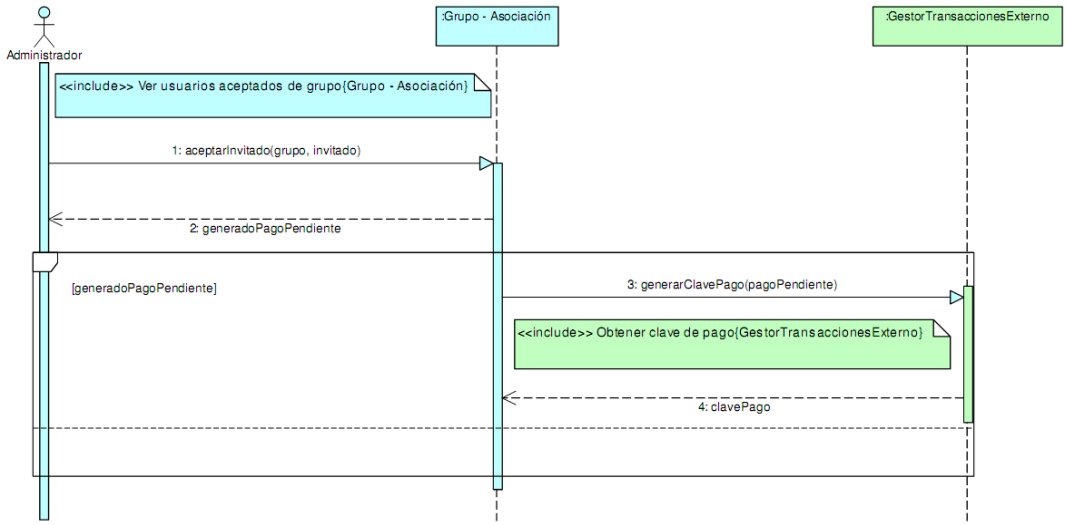
Postcondiciones:

El invitado es expulsado del grupo y tendrá el mismo acceso que cualquier otro usuario registrado que no participe en el proyecto.

invitado.solicitudAceptada -> oclIsUndefined() and
grupo.solicitudAceptada -> select(s | s.invitado = invitado) -> oclIsUndefined() and
grupo.invitadoAceptado -> excludes(invitado) and grupo.invitadoRechazado -> includes(invitado) and
grupo.conexiónChat.invitado -> excludes(invitado) and grupo.totalInvitados = grupo.totalInvitados - 1
invitado.estado = EstadoUsuario::Rechazado and infoUsuario.fecha = fechaActual()

Aceptar invitado

El invitado es aceptado como miembro. En caso de tener un pago pendiente tendrá el estado pendiente de aceptación hasta que no lo complete. Una vez completado o en caso de no tener ningún pago pendiente su estado pasa a ser aceptado. La diferencia entre los dos estados es que con estado pendiente de aceptación sigue teniendo los mismos permisos que un invitado, en cambio como aceptado consigue los permisos habituales de miembros como es el acceso al muro de mensajes y eventos.



Operación: aceptarInvitado(grupo: Grupo, invitado: Invitado)
Descripción: Acepta a un invitado como miembro del grupo
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el invitado en el sistema y ser un invitado aceptado en el grupo
Invitado.allInstances() -> exists(i | i = invitado) and grupo.invitadoAceptado -> includes(invitado)

El administrador lo es del propio grupo
grupo.creador.usuarioRegistrado = self

El total de miembros no menor que el máximo de invitados del grupo
grupo.totalMiembros < grupo.maxMiembros

El administrador del grupo no tiene bloqueado al usuario
grupo.creador.usuarioRegistrado.bloqueado -> excludes(self)

Postcondiciones:
El invitado pasa a ser miembro del grupo aumentando así los permisos dentro del grupo. En caso de tener un pago pendiente deberá completarlo antes de ser miembro con todos los permisos que conlleva.

invitado.solicitudAceptada -> oclIsUndefined() and
grupo.solicitudAceptada -> select(s | s.invitado = invitado) -> oclIsUndefined() and
grupo.invitadoAceptado -> excludes(invitado) and invitado.oclIsTypeOf(Miembro) and
grupo.miembroAceptado -> includes(invitado) and invitado.membresía = TipoUsuario::Miembro
grupo.totalInvitados = grupo.totalInvitados -1 and grupo.totalMiembros = grupo.totalMiembros + 1

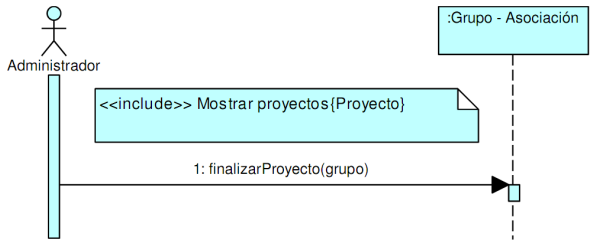
if invitado.pagoPendiente > 0 then
pagoPendiente.oclIsNew() and pagoPendiente.oclIsTypeOf(PagoPendiente) and
pagoPendiente.cantidad = infoUsuario.pagoPendiente and pagoPendiente.cuenta = grupo.cuenta and
pagoPendiente.divisa = grupo.divisa and pagoPendiente.fecha = fechaActual() and
pagoPendiente.identificador = PagoPendiente.allInstances().identificador -> max() +1
pagoPendiene.clavePago = GestorTransaccionesExterno::generarClavePago(pagoPendiente.cuenta,
pagoPendiente.divisa, pagoPendiente.cantidad)
pagoPendiente.grupo = grupo and pagoPendiente.usuarioRegistrado = invitado.usuarioRegistrado and
invitado.estado = EstadoUsuario::PendienteAceptación
else
invitado.estado = EstadoUsuario::Aceptado
endif

Operación: generarClavePago(pagoPendiente: PagoPendiente)
Descripción: Genera una clave de pago obtenida de la empresa gestora de transacciones
Precondiciones:
Debe existir el pago pendiente en el sistema
PagoPendiente.allInstances() -> exists(p | p = pagoPendiente)

Salida:
Retorna la clave de pago obtenida de la empresa gestora de transacciones

Finalizar proyecto

Finaliza el proyecto para indicar así que todos los objetivos han sido cumplidos y ya es una realidad, por tanto, no necesitará ya utilizar el proyecto publicado para desarrollarlo o pedir financiación. Aun así, el administrador y miembros pueden publicar actualizaciones sobre como el proyecto sigue creciendo fuera de la plataforma. Antes de finalizar definitivamente el proyecto todos los miembros, si los hay, deberán llegar a un acuerdo unánime.



Operación: finalizarProyecto(grupo: Grupo)
Descripción: Finaliza el proyecto para así indicar que todos los objetivos han sido cumplidos o interrumpe la finalización.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El administrador lo es del propio grupo
grupo.creador.usuarioRegistrado = self

El grupo tiene el estado Abierto, Cerrado o PendienteFinalización
grupo.estado = EstadoGrupo::Abierto or grupo.estado = EstadoGrupo::Cerrado or
grupo.estado = EstadoGrupo::PendienteFinalización

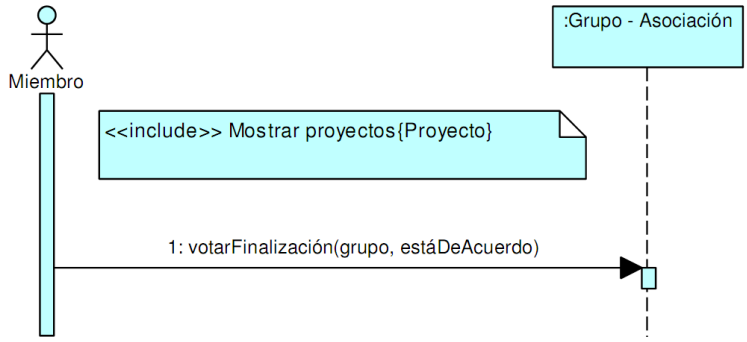
Postcondiciones:

El proyecto queda en estado pendiente de finalizar hasta que todos los miembros de este, en una votación unánime, indiquen que ya está finalizado. O si ya estaba pendiente de finalización, este se interrumpe volviendo a estar abierto. En caso de no haber miembros participando directamente finaliza el proyecto.

```
if grupo.miembroAceptado -> size() > 0 then
  if grupo.estado = EstadoGrupo::PendienteFinalización then
     grupo.estado = EstadoGrupo::Abierto
  else
    grupo.estado = EstadoGrupo::PendienteFinalización
  endif
else
   grupo.estado = EstadoGrupo::Finalizado
endif
```

Votar finalización proyecto

El miembro vota si está de acuerdo o no en finalizar el proyecto. Una vez que todos los miembros estén unánimemente de acuerdo con finalizar el proyecto, este pasa a estado finalizado.



Operación: votarFinalización(grupo: Grupo, estáDeAcuerdo: Boolean)
Descripción: Finaliza el proyecto para así indicar que todos los objetivos de éste están cumplidos o interrumpe la finalización.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El miembro lo es del propio grupo
grupo.miembroAceptado.usuarioRegistrado -> includes(self)

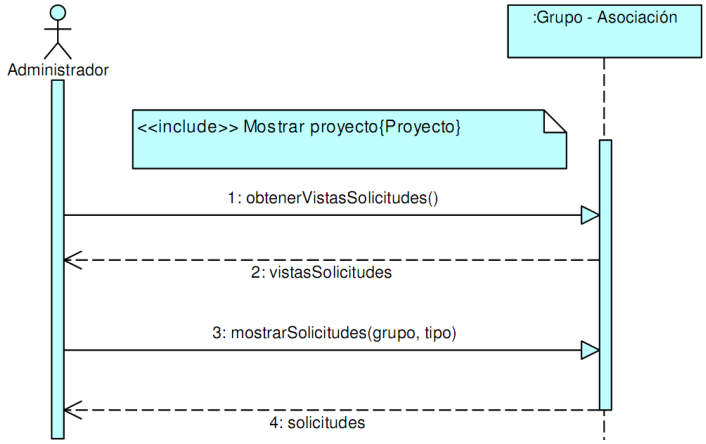
El grupo tiene el estado pendiente de finalización
grupo.estado = EstadoGrupo::PendienteFinalización

Postcondiciones:
El voto del miembro es guardado.

grupo.votoFinalización -> select(v | v.miembroAceptado.usuarioRegistrado = self).voto = estáDeAcuerdo

Mostrar solicitudes de grupo

Muestra el listado de solicitudes enviadas a un proyecto, clasificadas por vistas. Estas vistas muestran las solicitudes pendientes, aceptadas, rechazadas o todas.



Operación: obtenerVistasSolicitudes() : Set(String)
Descripción: Muestra el listado de vistas de solicitudes disponibles
Precondiciones: -
Salida:
let vistasSolicitudes : Set(String) -> oclIsUndefined() in
vistasSolicitudes -> includes('Todas') -> includes('Pendientes') -> includes('Aceptadas') -> includes('Rechazadas')

Operación: mostrarSolicitudes(grupo: Grupo, tipo: String): Set(TupleType(usuario: UsuarioRegistrado, solicitud: Solicitud))
Descripción: Muestra las solicitudes enviadas al grupo.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

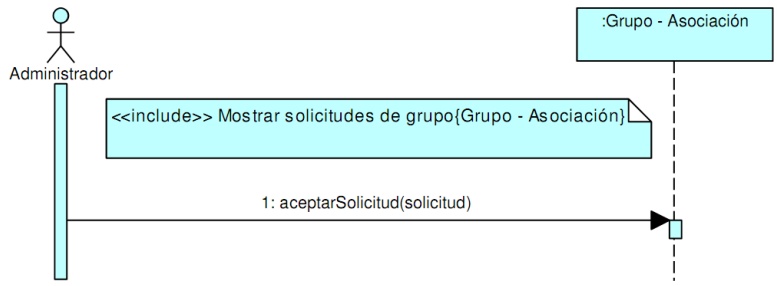
El administrador lo es del propio grupo
grupo.creador.usuarioRegistrado = self

Salida:
Genera el listado de solicitudes del grupo

```
let solicitudes : Set(TupleType(usuario: UsuarioRegistrado, solicitud: Solicitud)) -> isEmpty() in
if tipo = 'Todas' then
  grupo.solicitudPendiente
        -> iterate(s | solicitudes -> includes(Tuple(usuario = s.candidato.usuarioRegistrado, solicitud = s)))
  grupo.solicitudRechazada
        -> iterate(s | solicitudes -> includes(Tuple(usuario = s.candidato.usuarioRegistrado, solicitud = s)))
  grupo.solicitudAprobada
        -> iterate(s | solicitudes -> includes(Tuple(usuario = s.invitado.usuarioRegistrado, solicitud = s)))
elseif tipo = 'Pendientes' then
  grupo.solicitudPendiente
        -> iterate(s | solicitudes -> includes(Tuple(usuario = s.candidato.usuarioRegistrado, solicitud = s)))
elseif tipo = 'Rechazadas' then
  grupo.solicitudRechazada
        -> iterate(s | solicitudes -> includes(Tuple(usuario = s.candidato.usuarioRegistrado, solicitud = s)))
elseif tipo = 'Aceptadas' then
  grupo.solicitudAprobada
        -> iterate(s | solicitudes -> includes(Tuple(usuario = s.invitado.usuarioRegistrado, solicitud = s)))
endif
```

Aceptar solicitud

Acepta una solicitud de un candidato promoviéndolo así a invitado del proyecto.



Operación: aceptarSolicitud(solicitud: Solicitud)
Descripción: Acepta la solicitud de un candidato promoviéndolo a invitado.
Precondiciones:

Debe existir la solicitud en el sistema
Solicitud.allInstances() -> exists(s | s = solicitud)

El administrador lo es del propio grupo al que pertenece la solicitud
solicitud.grupo.creador.usuarioRegistrado = self

La solicitud está pendiente de aceptación o rechazo
not solicitud.oclIsTypeOf(SolicitudAceptada) and not solicitud.oclIsTypeOf(SolicitudRechazada)

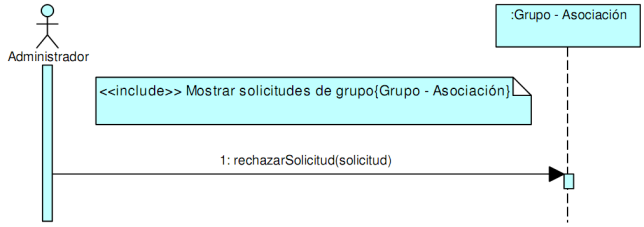
El total de invitados es menor que el máximo de invitados del grupo
grupo.totalInvitados < grupo.maxInvitados

Postcondiciones:
El candidato pasar a ser invitado del grupo y la solicitud pasa a ser aprobada

let candidato : infoUsuario = solicitud.candidato in
let grupo : Grupo = solicitud.grupo in
solicitud.oclIsType(SolicitudAprobada) and grupo.invitadoAceptado -> includes(candidato) and
grupo.candidatoAceptado -> excludes(candidato) and grupo.solicitudAprobada -> includes(solicitud) and
grupo.solicitudPendiente -> excludes(solicitud) and solicitud.invitado = candidato and
grupo.totalCandidatos = grupo.totalCandidatos -1 and grupo.totalInvitados = grupo.totalInvitados + 1
candidato.estado = EstadoUsuario::Aceptado and candidato.membresia = TipoUsuario::Invitado and
candidato.fecha = fechaActual() and candidato.oclIsTypeOf(Invitado)

Rechazar solicitud

La solicitud es rechazada y por tanto, el candidato es expulsado del proyecto por el administrador. Por tanto, deberá esperar 4 meses como máximo para volver a solicitar la entrada al proyecto.



Operación: rechazarSolicitud(solicitud: Solicitud)
Descripción: Rechaza la solicitud de un candidato y este será expulsado del grupo durante un máximo de 4 meses.
Precondiciones:

Debe existir la solicitud en el sistema
Solicitud.allInstances() -> exists(s | s = solicitud)

El administrador lo es del propio grupo al que pertenece la solicitud
solicitud.grupo.creador.usuarioRegistrado = self

La solicitud está pendiente de aceptación o rechazo
not solicitud.oclIsTypeOf(SolicitudAceptada) and not solicitud.oclIsTypeOf(SolicitudRechazada)

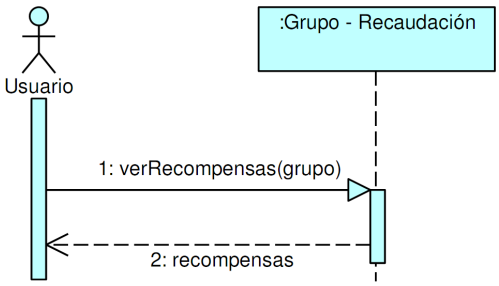
Postcondiciones:
El candidato es expulsado del grupo y la solicitud pasará a ser rechazada indicando al fecha actual como fecha de rechazo. Tendrá el mismo acceso que cualquier otro usuarioRegistrado que no participe en el proyecto.

let candidato : infoUsuario = solicitud.candidato in
let grupo : Grupo = solicitud.grupo in
solicitud.oclIsTypeOf(SolicitudRechazada) and solicitud.fechaRechazo = fechaActual() and
grupo.candidatoAceptado -> excludes(candidato) and solicitud.candidatoRechazado-> includes(candidato)
grupo.candidatoRechazado -> includes(candidato) and grupo.totalCandidatos = grupo.totalCandidatos - 1
candidato.estado = EstadoUsuario::Rechazado and candidato.fecha = fechaActual()

3.4.6. Diagramas de secuencia financiación de grupo

Ver recompensas

Muestra las recompensas disponibles para un proyecto en concreto. Cada recompensa se compone de no solo el nombre también una descripción, una meta, cantidad mínima para poder solicitarla al financiar, el número de solicitadas, el límite para este número de solicitadas y el tiempo estimado en semanas que tarda en estar preparada para enviar.



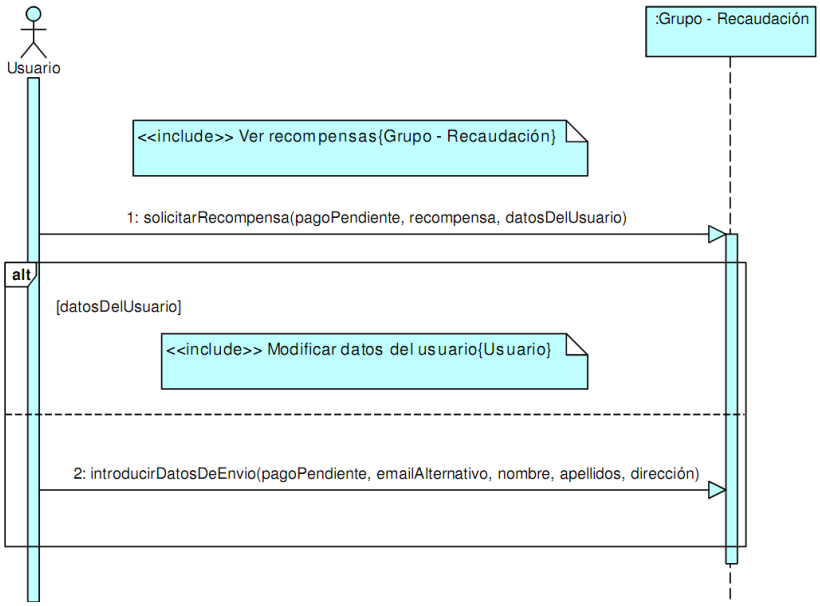
Operación: verRecompensas(grupo: Grupo): Set(Recompensa)
Descripción: Genera un listado de recompensas disponibles del grupo
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

La recaudación y el sistema de recompensas del grupo deben estar activados
grupo.recaudaciónActiva = true and
grupo.sistemaRecompensasActivo = true

Salida:
Retorna una colección de recompensas del grupo

Solicitar recompensa

Permite solicitar una recompensa cada vez que se financia un proyecto. Las recompensas disponibles dependerán de la cantidad aportada. El usuario que financia tiene la posibilidad de no pedir ninguna recompensa o pedir una. En este último caso será requerido que introduzca los datos de contacto y dirección de envío. Estos datos pueden ser tomados directamente del perfil del usuario o puede indicar nuevos. En el caso de un usuario anónimo solo tiene la posibilidad de indicar nuevos datos.



Operación: solicitarRecompensa(pagoPendiente: PagoPendiente, recompensa: Recompensa, datosDelUsuario: boolean)
Descripción: Crea una recompensa pendiente en caso de que sea solicitada
Precondiciones:
Debe existir el pagoPendiente en el sistema y estar asociado al usuario
PagoPendiente.allInstances() -> exists(p | p = pagoPendiente) and pagoPendiente.usuario = self

Debe existir la recompensa en el sistema y estar asociado al mismo grupo que el pago pendiente
Recompensa.allInstances() -> exists(r | r = recompensa) and recompensa.grupo = pagoPendiente.grupo

El recaudación y el sistema de recompensas del grupo al que pertenece la recompensa deben estar activados
recompensa.grupo.recaudaciónActiva = true and recompensa.grupo.sistemaRecompensasActivo = true

La recompensa no puede estar vacía, ni tener una meta superior a la cantidad aportada por el usuario ni tener ninguna recompensa disponible ni estar desactivada
recompensa -> notEmpty() and recompensa.meta <= pagoPendiente.cantidad and
(recompensa.limiteSolicitadas = 0 or recompensa.solicitadas < recompensa.limiteSolicitadas) and recompensa.activada

Postcondiciones:
Se crea una nueva recompensaSolicitada que se asocia al usuario y al grupo y se actualiza el número de recompensas solicitadas. Además se guarda el origen de los datos siendo posibles del perfil del usuario o guardados en la propia recompensa solicitada.
let rs : RecompensaSolicitada -> oclIsUndefined() in
rs.oclIsNew() and rs.oclIsTypeOf(RecompensaSolicitada) and
rs.identificador = RecompensaSolicitada.allInstances().identificador -> max() + 1 and
rs.recompensa = recompensa and rs.usuarioRegistrado = self and rs. pagoPendiente.recompensaSolicitada = rs and
rs.fecha = fechaActual() and rs.estado = EstadoRecompensa::PendienteDePago and
recompensa.solicitadas = recompensa.solicitadas + 1

if datosDelUsuario and not usuario.oclIsTypeOf(UsuarioAnónimo) then
rs.origenDatosEnvio = OrigenDatosEnvioRecompensa::PerfilUsuario
else
rs.origenDatosEnvio = OrigenDatosEnvioRecompensa::RecompensaSolicitada
endif

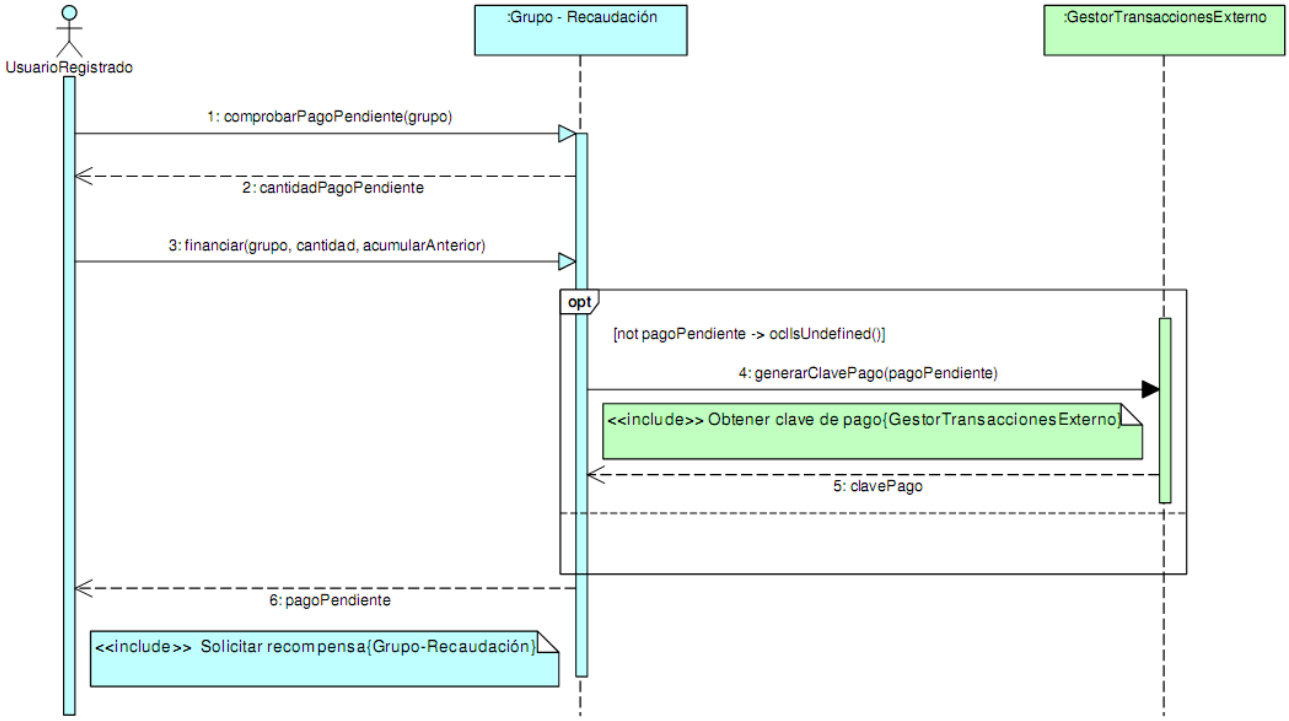
Operación: introducirDatosDeEnvio(pagoPendiente: PagoPendiente, emailAlternativo: String, nombre: String, apellidos: String, dirección: String)
Descripción: Introduce los datos de envío en la propia recompensa solicitada.
Precondiciones:
Se debe haber seleccionado una recompensa para el pago pendiente.
not pagoPendiente.recompensaSolicitada -> oclIsUndefined()

Postcondiciones:
Se guardan los datos de envío en la recompensa solicitada.
pagoPendiente.recompensaSolicitada.datosEnvio = Tuple{email = emailAlternativo, nombre = nombre, apellidos = apellidos, dirección = dirección}

Financiar

Financia un proyecto, creando un pago pendiente y una clave de pago, clave necesaria para completar el pago con el gestor de transacciones. El usuario, antes de financiar en caso de haber un pago pendiente anterior no completado, es informado si quiere acumular la cantidad del pago pendiente a la nueva aportación o si quiere hacer una nueva aportación eliminando ese pago pendiente. En caso de ser invitado o candidato del proyecto, no se crea ningún pago pendiente sino que solo se anota la cantidad para crear ese pago pendiente cuando llegue a ser miembro. Hemos seguido este proceso con los candidatos e invitados porque es más fácil que sean expulsados del proyecto que un miembro, el cual ya ha pasado más filtros. Al no ser el miembro expulsado tan fácilmente es menos probable que tenga que crearse una devolución de los pagos realizados por este cuando es expulsado.

Con cada aportación o financiación el usuario puede pedir una recompensa asociada a esa aportación. La cual una vez completado el pago sería recibida pasado el tiempo estimado en semanas que indica la recompensa.



Operación: ComprobarPagoPendiente(grupo: Grupo): Integer
Descripción: Comprueba si habia un pago pendiente y devuelve la cantidad del pago pendiente.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

La recaudación del grupo debe estar activada
grupo.recaudaciónActiva = true

Salida:
Comprueba si habia un pago pendiente y devuelve la cantidad del pago pendiente.

let cantidadPagoPendiente : Integer = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self).pagoPendiente

Operación: generarClavePago(pagoPendiente: PagoPendiente)
Descripción: Genera una clave de pago obtenida de la empresa gestora de transacciones
Precondiciones:
Debe existir el pago pendiente en el sistema
PagoPendiente.allInstances() -> exists(p | p = pagoPendiente)

Salida:
Retorna la clave de pago obtenida de la empresa gestora de transacciones

Operación: Financiar(grupo: Grupo, cantidad: Integer, acumularAnterior: Boolean): PagoPendiente
Descripción: Financia el proyecto con indicando cuantas unidades monetarías va a aportar.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

La recaudación del grupo debe estar activada

```
grupo.recaudaciónActiva = true
```

La cantidad no está vacía y es mayor o igual que 1

```
not cantidad -> oclIsUndefined() and cantidad >= 1
```

Postcondiciones:

Si el usuario participa en el proyecto como invitado o candidato, se guardará la cantidad indicada sin crear ningún pago pendiente hasta que no llegue a ser miembro. Para el resto de usuario será creado un pago pendiente.

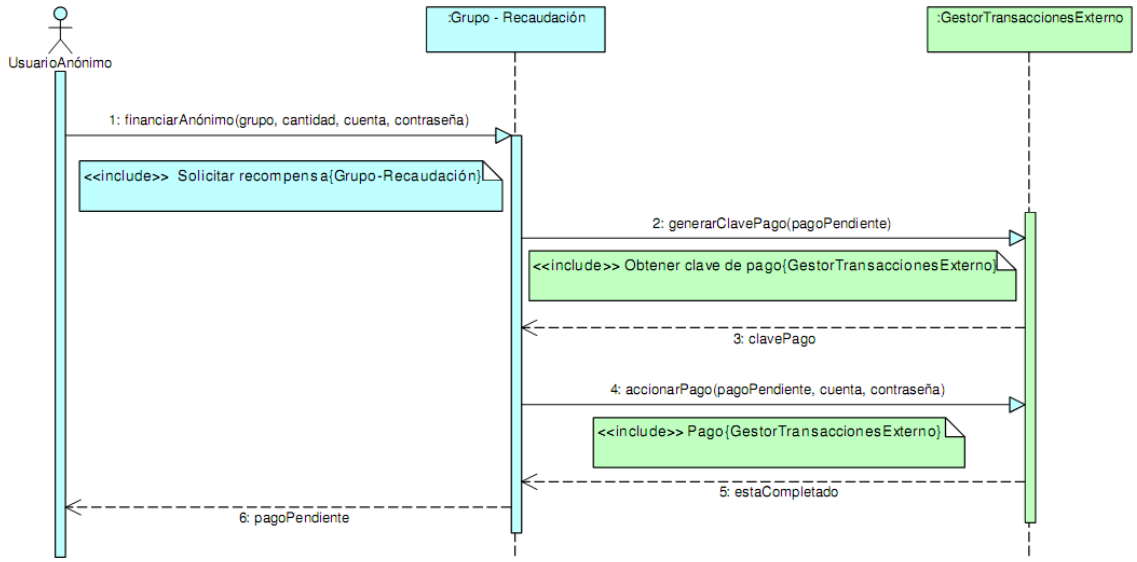
```
let t : PagoPendiente -> oclIsUndefined() in
if self.infoUsuario.grupo -> includes(grupo) then
  let info: infoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
  if info.membresía = TipoUsuario::Administrador  or info.membresía = TipoUsuario::Miembro or
    info.membresía = TipoUsuario::NoParticipa then
      if acumularAnterior then
        info.pagoPendiente = info.pagoPendiente + cantidad
      else
        info.pagoPendiente = cantidad
      endif

      let pagoPendiente : PagoPendiente = grupo.pagoPendiente -> select(p | p.usuarioRegistrado = self) in
      if pagoPendiente -> size() = 1 then
        PagoPendiente.allInstances -> excludes(pagoPendiente)
      endif
      t.oclIsNew() and t.oclIsTypeOf(PagoPendiente) and
       t.identificador = PagoPendiente.allInstances().identificador -> max() + 1 and
      t.usuarioRegistrado = self and t.grupo = grupo and t.tipo = TipoPagoPendiente::PagoDeParticipante and
       intentoCompletar = false and t.divisa = grupo.divisa and t.cuenta = grupo.cuenta and t.cantidad = cantidad and
       t.fecha = fechaActual() and
      t.clavePago = GestorTransaccionesExterno::generarClavePago(t.cuenta, t.divisa, t.cantidad)
  elseif info.membership = TipoUsuario::Invitado or info.membership = TipoUsuario::Candidato then
      if acumularAnterior then
        info.pagoPendiente = info.pagoPendiente + cantidad
      else
        info.pagoPendiente = cantidad
      endif
  endif
else
  info.oclIsNew() and info.oclIsTypeOf(infoUsuario) and info.inversión = 0 and info.devoluciónPendiente = 0 and
  info.destacaProyecto = false and info.rol = '' and info.membresía = TipoUsuario::NoParticipa and
  info.estado = EstadoUsuario::Aceptado and info.fecha = fechaActual() and info.grupo = grupo and
  info.usuarioRegistrado = self
  if acumularAnterior then
    info.pagoPendiente = info.pagoPendiente + cantidad
  else
    info.pagoPendiente = cantidad
  endif

  let pagoPendiente : PagoPendiente = grupo.pagoPendiente -> select(p | p.usuarioRegistrado = self) in
  if pagoPendiente -> size() = 1 then
    PagoPendiente.allInstances -> excludes(pagoPendiente)
  endif
  t.oclIsNew() and t.oclIsTypeOf(PagoPendiente) and
  t.identificador = PagoPendiente.allInstances().identificador -> max() + 1 and t.usuarioRegistrado = self and
  t.grupo = grupo and t.tipo = TipoPagoPendiente::PagoDeNoParticipante and intentoCompletar = false and
  t.divisa = grupo.divisa and t.cuenta = grupo.cuenta and t.cantidad = cantidad and t.fecha = fechaActual() and
  t.clavePago = GestorTransaccionesExterno::generarClavePago(t.cuenta, t.divisa, t.cantidad)
endif
```

Salida:

```
let pagoPendiente : PagoPendiente = t
```

Financiar anónimo

El usuario anónimo financia el proyecto y debe completar el pago inmediatamente introduciendo la cuenta y contraseña de su cuenta en el gestor de transacciones. Para ello antes se genera la clave de pago y entonces el sistema se comunica con el gestor para completar el pago. En caso de completarse correctamente y haber solicitada una recompensa, la recibiría pasado el tiempo estimado en semanas que indica la recompensa.



Operación: FinanciarAnónimo(grupo: Grupo, cantidad: Integer, cuenta: String, contraseña: String): PagoPendiente
Descripción: Financia el proyecto indicando cuantas unidades monetarías va a aportar al proyecto.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

La recaudación del grupo debe estar activada

```
grupo.recaudaciónActiva = true
```

La cantidad no está vacía y es mayor o igual que 1

```
not cantidad -> oclIsUndefined() and cantidad >= 1
```

Postcondiciones:

Crea un pagoPendiente y lo asocia a el usuarioAnónimo.

```
let t : PagoPendiente -> oclIsUndefined() in
t.oclIsNew() and t.oclIsTypeOf(PagoPendiente) and
t.identificador = PagoPendiente.allInstances.identificador -> max() + 1 and t.usuarioAnónimo = self and
t.grupo = grupo and t.tipo = TipoPagoPendiente::PagoDeNoParticipante and intentoCompletar = false and
t.divisa = grupo.divisa and t.cuenta = grupo.cuenta and t.cantidad = cantidad and t.fecha = fechaActual()
t.clavePago = GestorTransaccionesExterno::generarClavePago(t.cuenta, t.divisa, t.cantidad)

let estaCompletado : Boolean = GestorTransaccionesExterno::accionarPago(t, cuenta, contraseña) in
if estaCompletado then
  t.oclIsTypeOf(PagoCompletado)
  if  t.recompensaSolicitada <> null then
     t.recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado
  endif
  grupo.recaudadoTotal = grupo.recaudadoTotal + cantidad
  grupo.totalFinanciadores = grupo.totalFinanciadores + 1
endif
```

Operación: generarClavePago(pagoPendiente: PagoPendiente)
Descripción: Genera una clave de pago obtenida de la empresa gestora de transacciones
Precondiciones:
Debe existir el pago pendiente en el sistema
PagoPendiente.allInstances() -> exists(p | p = pagoPendiente)

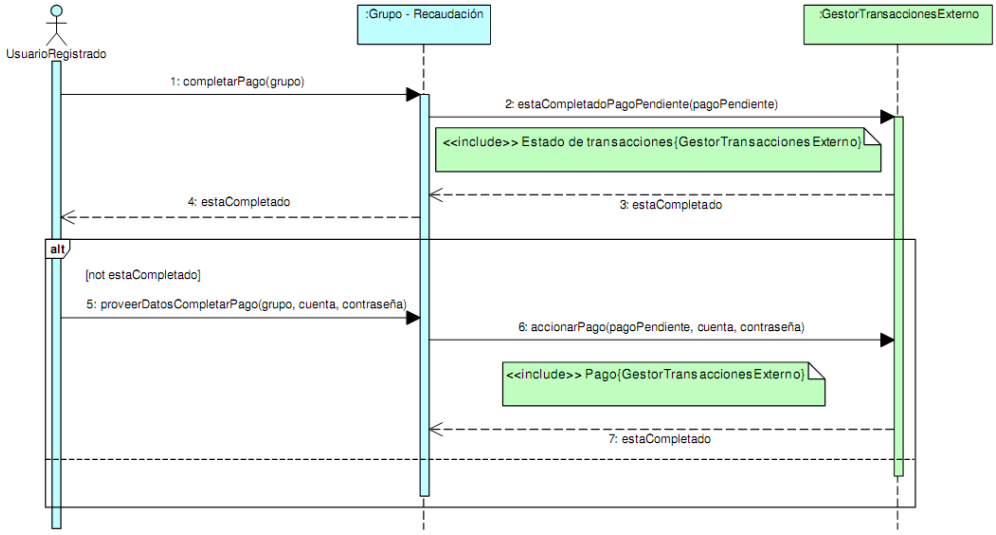
Salida:
Retorna la clave de pago obtenida de la empresa gestora de transacciones

Operación: accionarPago(pagoPendiente: pagoPendiente, cuenta: String, contraseña: String): Boolean
Descripción: Completa el pago comunicándose con la empresa gestora de transacciones externa y proporcionándole una cuenta y contraseña como usuario del propio gestor.
Precondiciones:
Debe existir el pago pendiente en el sistema
PagoPendiente.allInstances() -> includes(PagoPendiente)

Salida:
Comunica a la empresa gestora de transacciones la información del pago pendiente a completar y esta devuelve una respuesta indicando si se ha completado

Completar pago

Completa el pago pendiente con el proyecto. Primero comprueba que en un intento anterior no esté ya completado si no lo estuviera le solicita la cuenta y contraseña de su cuenta con el gestor de transacciones. Una vez comprobado que ya está completado el pago, este pasa a la recaudación del proyecto y el pago pendiente es transformado en completado, actualizando así otros atributos necesarios. La especificación de la operación accionarPago es la misma que la del diagrama anterior de financiar Anónimo.



Operación: estaCompletadoPagoPendiente(pagoPendiente: PagoPendiente): Boolean
Descripción: Comprueba si el pago pendiente está completado
Precondiciones:
Debe existir el grupo en el sistema
PagoPendiente.allInstances() -> exists(p | p = pagoPendiente)

Existe un pago pendiente del usuario con el proyecto
self.pagoPendiente -> select(p | p = pagoPendiente) -> notEmpty()

La recaudación debe estar activada
pagoPendiente.grupo.recaudaciónActiva = true

Salida:
Retorna un valor booleano indicando si el pago pendiente está completado consultando con el gestor de transacciones

Operación: completarPago(grupo: Grupo)
Descripción: Completa el pago pendiente con el proyecto
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

Existe un pago pendiente del usuario con el proyecto

```
self.pagoPendiente -> select(p | p.grupo = grupo) -> notEmpty()
```

La recaudación debe estar activada

```
grupo.recaudaciónActiva = true
```

Postcondiciones:

Comprueba si está completado el pago de un intento anterior de completarlo. Si lo está y tenía alguna recompensa asociada está pasará a estado Pago completado, además el pago pasará a formar parte de la recaudación del proyecto. Si es miembro del proyecto pero está pendiente de aceptación por tener un pago pendiente a completar, podrá a partir de este momento participar como miembro de pleno derecho.

```
let pagoPendiente : PagoPendiente = self.pagoPendiente -> select(p | p.grupo = grupo) in
let estaCompletado: Boolean = GestorTransaccionesExterno::estaCompletadoPagoPendiente(pagoPendiente) in

if estaCompletado then
  pagoPendiente.oclIsTypeOf(PagoCompletado)
  if not pagoPendiente.recompensaSolicitada -> oclIsUndefined then
    pagoPendiente.recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado
  endif
  grupo.recaudadoTotal = grupo.recaudadoTotal + pagoPendiente.cantidad
  if grupo.pagoCompletado.usuarioRegistrado -> excludes(self) then
    grupo.totalFinanciadores = grupo.totalFinanciadores + 1
  endif
  let infoUsuario : InfoUsuario = self.infoUsuario -> select(u | u.grupo = grupo) in
  infoUsuario.pagoPendiente = 0 and infoUsuario.inversión = infoUsuario.inversion + pagoPendiente.cantidad
  if infoUsuario.membresia = TipoUsuario::Miembro and infoUsuario.estado = EstadoUsuario::PendienteAceptación then
    infoUsuario.estado = EstadoUsuario::Aceptado
  endif
endif
```

Operación: proveerDatosCompletarPago(grupo: Grupo, cuenta: String, contraseña: String)
Descripción: Completa el pago pendiente con el proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Existe un pago pendiente del usuario con el proyecto
self.pagoPendiente -> select(p | p.grupo = grupo) -> notEmpty()

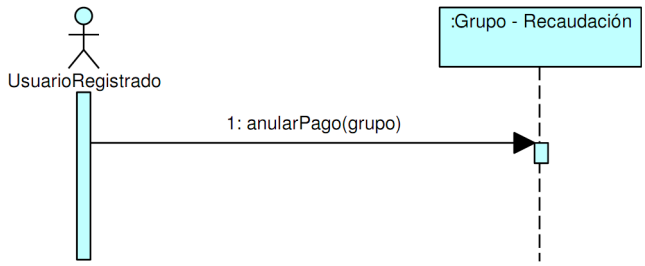
La recaudación debe estar activada
grupo.recaudaciónActiva = true

Postcondiciones:
Si completa el pago y tenía alguna recompensa asociada está pasará a estado Pago completado, además el pago pasará a formar parte de la recaudación del proyecto. Si es miembro del proyecto pero está pendiente de aceptación por tener un pago pendiente a completar, podrá a partir de este momento participar como miembro de pleno derecho.

```
let pagoPendiente : PagoPendiente = self.pagoPendiente -> select(p | p.grupo = grupo) in
let estaCompletado: Boolean = GestorTransaccionesExterno::accionarPago(pagoPendiente, cuenta, contraseña)

if estaCompletado then
  pagoPendiente.oclIsTypeOf(PagoCompletado)
  if not pagoPendiente.recompensaSolicitada -> oclIsUndefined then
    pagoPendiente.recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado
  endif
  grupo.recaudadoTotal = grupo.recaudadoTotal + pagoPendiente.cantidad
  if grupo.pagoCompletado.usuarioRegistrado -> excludes(self) then
    grupo.totalFinanciadores = grupo.totalFinanciadores + 1
  endif
  let infoUsuario : InfoUsuario = self.infoUsuario -> select(u | u.grupo = grupo) in
  infoUsuario.pagoPendiente = 0 and infoUsuario.inversión = infoUsuario.inversion + pagoPendiente.cantidad
  if infoUsuario.membresia = TipoUsuario::Miembro and infoUsuario.estado = EstadoUsuario::PendienteAceptación then
    infoUsuario.estado = EstadoUsuario::Aceptado
  endif
endif
```

Anular pago

Anula un pago pendiente con el proyecto. Solo será posible para usuarios registrados que no participen en este, candidatos, miembros no pendientes de aceptación y administrador. Hemos decidido que para los invitados y miembros pendientes de aceptación no es posible anular el pago pendiente porque en el momento en que el administrador del proyecto acepta una solicitud y está promete una inversión inicial, debe mantenerse o aumentar hasta poder completar el pago al llegar a ser miembro. Por tanto, hasta que el miembro pendiente de aceptación no complete el pago, este pago pendiente no podrá desaparecer, a no ser que antes el usuario sea expulsado o él mismo salga del proyecto.



Operación: anularPago(grupo: Grupo)
Descripción: Anula un pago pendiente hacia un proyecto
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El usuario tiene un pago pendiente con el grupo

```
grupo.pagoPendiente -> select(p | p.usuarioRegistrado = self) -> notEmpty()
```

El usuario no es invitado del grupo ni miembro con estado pendiente de aceptación

```
let infoUsuario : infoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
grupo.invitadoAceptado -> excludes(infoUsuario) and
not(grupo.miembroAceptado -> includes(infoUsuario) and infoUsuario.estado = EstadoUsuario::PendienteAceptación)
```

Postcondiciones:

El pago pendiente es eliminado y la información de pago pendiente entre el usuario y el proyecto es actualizado.

```
let infoUsuario : infoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
infoUsuario.pagoPendiente = 0

let recompensaSolicitada : RecompensaSolicitada =
        grupo.recompensa.recompensaSolicitada -> select(r | r.usuarioRegistrado = self and
                                                            r.estado = EstadoRecompensa::PendienteDePago) in
if not recompensaSolicitada -> oclIsUndefined() then
  recompensaSolicitada.recompensa.solicitadas = recompensaSolicitada.recompensa.solicitadas - 1
  if not recompensaSolicitada.recompensa.activada and recompensaSolicitada.recompensa.solicitadas = 0 then
    recompensaSolicitada.recompensa.grupo -> oclIsUndefined and
    Recompensa.allInstances() -> excludes(recompensaSolicitada.recompensa)
  endif
  recompensaSolicitada.recompensa -> oclIsUndefined and
  RecompensaSolicitada.allInstances() -> excludes(recompensaSolicitada)
endif
if grupo.candidatoAceptado -> excludes(infoUsuario) then
   grupo.pagoPendiente -> select(p | p.usuarioRegistrado = self) -> isEmpty()
endif
```

4. Desarrollo

4.1. Análisis de gestores de contenidos o CMS

Durante el análisis y comparativa de diversos CMS, hemos encontrado que existe una gran variedad de éstos. Sin embargo, los más populares son Wordpress, Drupal y Joomla.

Los tres están escritos en PHP, ampliamente utilizado en servidores de todo el mundo. Aunque existen otros lenguajes como Python, Ruby, Java, Perl, etc. utilizados también en otros CMS como base de su código.

Hemos puesto la atención sobre estos tres especialmente no solo por la popularidad de uso, sino también por su gratuidad, específicamente del núcleo(compuesto por los módulos de la instalación inicial del gestor).

De forma adicional, se ha analizado uno escrito en Java llamado Magnolia.

La experiencia previa en la creación de plataformas web es muy pequeña y por tanto, empezaremos describiendo qué experiencia han tenido otros usuarios que hayan lidiado con estos gestores.

Generalmente, no solo tienen la ventaja de ser código abierto y la gratuidad de sus núcleos y la mayoría de sus módulos y temas(compuestos por plantillas y archivos CSS que forman la interfaz), sino que además, disponen de una gran comunidad de usuarios.

Estas comunidades de usuarios crean y mantienen una gran de cantidad de módulos que extienden las funcionalidades básicas de estos gestores.

La gran diferencia no solo en términos de popularidad y uso en soluciones finales radica sobretodo en la curva de aprendizaje, siendo la menos pronunciada Wordpress, algo más pronunciada Joomla y prácticamente exponencial para Drupal.

Drupal parece ser la última opción a escoger, para alguien sin experiencia previa. Sin embargo, debemos tener en cuenta el punto de vista desde el que se mira. Todo gestor o diseñador web no acostumbrado a trabajar en código encontrará extremadamente difícil adecuarlo a sus necesidades ya que está enfocada a la extensibilidad a través del código.

Sin embargo, un programador acometerá esta tarea con mucha más facilidad. Por tanto, debemos diferenciar si esta curva de aprendizaje es para desarrollar una web que gestione únicamente contenidos, tales como blogs, webs de noticias, foros, que es el cometido básico de los gestores, o que vaya más allá y permita además otros tipos de interacción del usuario con la plataforma.

Para nuestra plataforma necesitábamos ir más allá y poder gestionar interacciones, tales como las relaciones entre cada proyecto y el usuario y la funcionalidades permitidas en cada proyecto que de estas se deriven

Por tanto, aunque aprender a utilizar Joomla y Wordpress como simple gestor de contenidos es tarea fácil, a la hora de desarrollar nuevos módulos no lo es tanto. Al contrario que Drupal, el cuál permite fácilmente crear nuevos módulos y que interaccionen con los ya existentes sin realizar cambios estos últimos.

La gran comunidad de usuarios de cada uno y la enorme cantidad de extensiones que desarrollan, en un principio, deberían ser suficientes para las necesidades de esta plataforma. Sin embargo, no fue así, tras la búsqueda de las extensiones necesarias, no fueron encontradas todas o eran de pago o simplemente necesitaba realizarse un gran trabajo desactivando funcionalidades y otorgando nuevas.

Así que la decisión tomada fue crearlas desde cero y así controlar con total precisión todas y cada una de ellas y que se adapten a los requerimientos del proyecto.

El CMS que con más facilidad permitía crear extensiones es Drupal.

El anteriormente mencionado Magnolia daba como ventaja el hecho que estuviese basado en Java y no PHP, lo que consideraba como ventaja por la carencia de experiencia previa en este último. La gran

desventaja es la aparente inexistencia de comunidad de usuarios que desarrollen extensiones y la pobre documentación existente, esto sobretodo lo hacía totalmente inadecuado para utilizarlo. Drupal en cambio dispone de muy buena documentación, además de foros y webs centradas en este gestor que la complementan facilitando enormemente el desarrollo.

4.2. Introducción al gestor de contenidos Drupal

Drupal es un sistema de gestión web o CMS(acrónimo de Content Management System) compuesto por un determinado número de módulos escritos en PHP que componen su núcleo y apoyado por una base de datos relacional compatible con mysql, postgresql o sqlite.

Administra desde procesos a más bajo nivel tales como las conexiones entre servidor y clientes, hasta la gestión de usuarios, sesiones, gestión de contenido, gestión de distribución de contenido. Además gestiona otros componentes necesarios en la web como menús o bloques y la configuración de apariencia además de otros aspectos más técnicos como el mantenimiento y desarrollo web.

El núcleo de Drupal está repartido en varios directorios, uno para los módulos principales y otro para archivos de código en PHP con funcionalidades comunes. Además dispone de un directorio de scripts para realizar ciertas tareas, como indexación, mantenimiento, etc, otro directorio llamado misc con recursos varios tales como imágenes y por último, dispone de otra para los temas incluidos por defecto.

Por último, el directorio que contiene el sitio web con los módulos y temas añadidos por el administrador es sites. La separación entre el núcleo y el sitio web en si es debido a que se quiere facilitar las actualizaciones y la movilidad del sitio web a los servidores. Además teniendo concentrado bajo un mismo directorio los datos importantes, facilitará también las copias de seguridad.

Los componentes principales de Drupal son los módulos, nodos y hooks.

Los módulos son los componentes hechos en código PHP que sirven para extender las funcionalidades del sistema Drupal. Los nodos son pedazos de contenido principalmente compuestos de un título, un cuerpo y comentarios componen el sitio web. Por último, los hooks son funciones especiales que son ejecutadas en el momento adecuado dentro de los procesos de la web. Para conseguir esto son invocados en un orden y momento específicos que están determinados por el nombre de la propia función. Para más información ver Apéndice III – Guía de utilización Drupal.

4.3. Módulos implementados

Siguiendo el esquema de clases de la etapa de especificación en esta sección han sido delimitadas las funcionalidades en varios módulos. Estos han sido implementados en el entorno Drupal con código propio, utilizando las herramientas que proporciona el CMS para acoplar los nuevos módulos

Proyecto. Define el nodo proyecto, además de albergar todas las funciones necesarias para gestionar tanto la información del nodo como la presentación.

Grupo. Encargado de extender las funcionalidades de proyecto. Gestiona la asociación de usuarios al proyecto, las vías de comunicación y la financiación entre otros aspectos.

Usuario. Extiende las funcionalidades que ofrece por defecto el módulo user de Drupal.

Solicitud. Gestiona las solicitudes de entrada al proyecto y su presentación.

Evento. Define el nodo evento, además de albergar todas las funciones necesarias para gestionar tanto

la información del nodo como la presentación.
Chat. Gestiona la creación y distribución de los mensajes de chat entre los usuarios conectados al proyecto.

Muro. Define el nodo muro y sus comentarios, además de albergar todas las funciones necesarias para gestionar tanto la información del nodo como la presentación.

Correo. Gestiona el envío y visualización de los correos de cada usuario.

Categorías. Aun no apareciendo como una clase en la especificación, alberga las categorías utilizadas en el proyecto. Gestiona además las categorías sugeridas, que todo usuario administrador del proyecto puede utilizar en caso de que ninguna de las ofrecidas se ajuste a su proyecto.

Utils. Gestiona funciones y aspectos comunes de los módulos anteriores y que no encajan únicamente en ninguno de ellos.

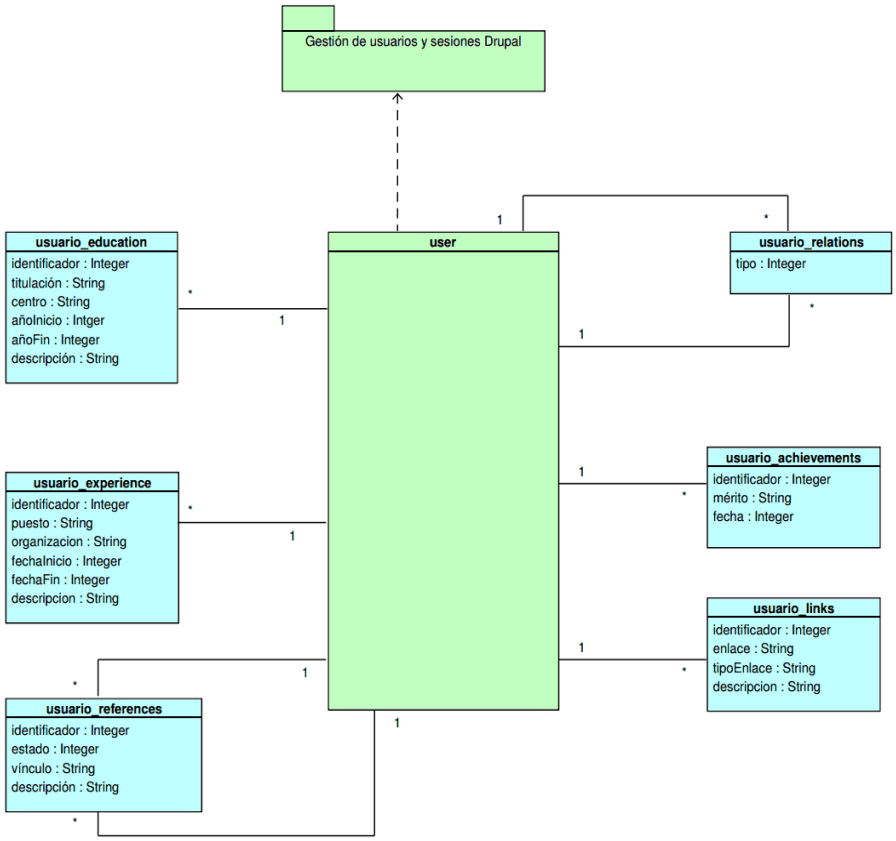
4.4. Estructura de la Base de Datos

El gestor de contenidos Drupal proporciona de forma inicial y por defecto unas 70 tablas para gestionar las funcionalidades principales y básicas. Entre ellas se encuentran tablas para gestionar la configuración del sistema, el cache, los bloques, sesiones, usuarios, nodos, incluyendo revisiones, comentarios y campos, indexación de contenido, variables, procesos por lotes, url amigables, menús, rutas y traducción.

A continuación exponemos el esquema de base de datos para la plataforma, la cual se ha dividido por claridad en gran medida según la estructura de módulos del sitio.

Hay un esquema de colores, como anteriormente indicado, las tablas o partes que gestiona Drupal están coloreadas en verde y para las tablas ya definidas con anterioridad en amarillo. Para así centrar la atención en las tablas en azul que van a ser las que se definirán.

4.4.1. Estructura base de datos del módulo Usuario



La tabla user pertenece al sistema Drupal y guarda los datos de cada usuario registrado, además de tener dos usuarios especiales por defecto. El primero tiene el identificador 0 y es asignado para todos los usuarios anónimos, lo que les diferencia es la sesión asignada. El segundo usuario especial es el superusuario o administrador del sistema con identificador 1 y que se encarga de gestionar toda la plataforma. A continuación son descritas las tablas con sus atributos y claves primarias.

usuario_relations

Esta tabla gestiona las relaciones entre los usuarios que pueden ser un bloqueo de un usuario a otro o que un usuario destaque a otro. Ya que no puede un usuario a la vez bloquear y destacar al mismo usuario, por eficiencia, han sido fusionadas las asociaciones lista de bloqueados y lista de destacados de la especificación de clases en esta tabla.

Clave primaria. (userMarker.user, userMarked.user)
Atributos.

- tipo – Valor entero que identifica el tipo de marca si de bloqueo o destacado.

Atributos foráneos.

- userMarker.user – Valor entero que identifica al usuario que bloquea o destaca a otro.
- userMarked.user – Valor entero que identifica al usuario que ha sido bloqueado o destacado.

usuario_achievements

Esta tabla gestiona los méritos que el propio usuario añade para ser mostrados en su perfil.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el mérito.
- mérito – Cadena que contiene el nombre del mérito.
- fecha – Valor entero que contiene el timestamp de la fecha en que se recibió el mérito.

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario.

usuario_links

Esta tabla gestiona los enlaces que el propio usuario añade para ser mostrados en su perfil.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el mérito.
- enlace – Cadena que contiene el nombre del mérito.
- tipoEnlace – Cadena que contiene el tipo de enlace, ya se blog, Facebook, twitter, G+, Linkedin, etc.
- descripción - Cadena que contiene la descripción del enlace(opcional)

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario.

usuario_education

Esta tabla gestiona las formaciones académicas que el propio usuario añade para ser mostrados en su perfil.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el mérito.
- titulación– Cadena que contiene la titulación de la formación.
- centro – Cadena que contiene el centro donde se tituló.
- añoInicio – Valor entero que contiene el año en que inició la formación.
- añoFin – Valor entero que contiene el año en que acabó la formación.
- descripción - Cadena que contiene la descripción de la formación(opcional)

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario.

usuario_experience

Esta tabla gestiona las experiencias profesionales que el propio usuario añade para ser mostrados en su perfil.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el mérito.
- puesto– Cadena que contiene el puesto que ocupó en la empresa u organización.
- organización – Cadena que contiene el nombre de la organización donde trabajó o colaboró.
- fechaInicio – Valor entero que contiene el timestamp de la fecha en que inició la experiencia.
- fechaFin – Valor entero que contiene el timestamp de la fecha en que acabó la experiencia.
- descripción - Cadena que contiene la descripción de la experiencia(opcional)

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario.

usuario_references

Esta tabla gestiona las experiencias profesionales que el propio usuario añade para ser mostrados en su perfil.

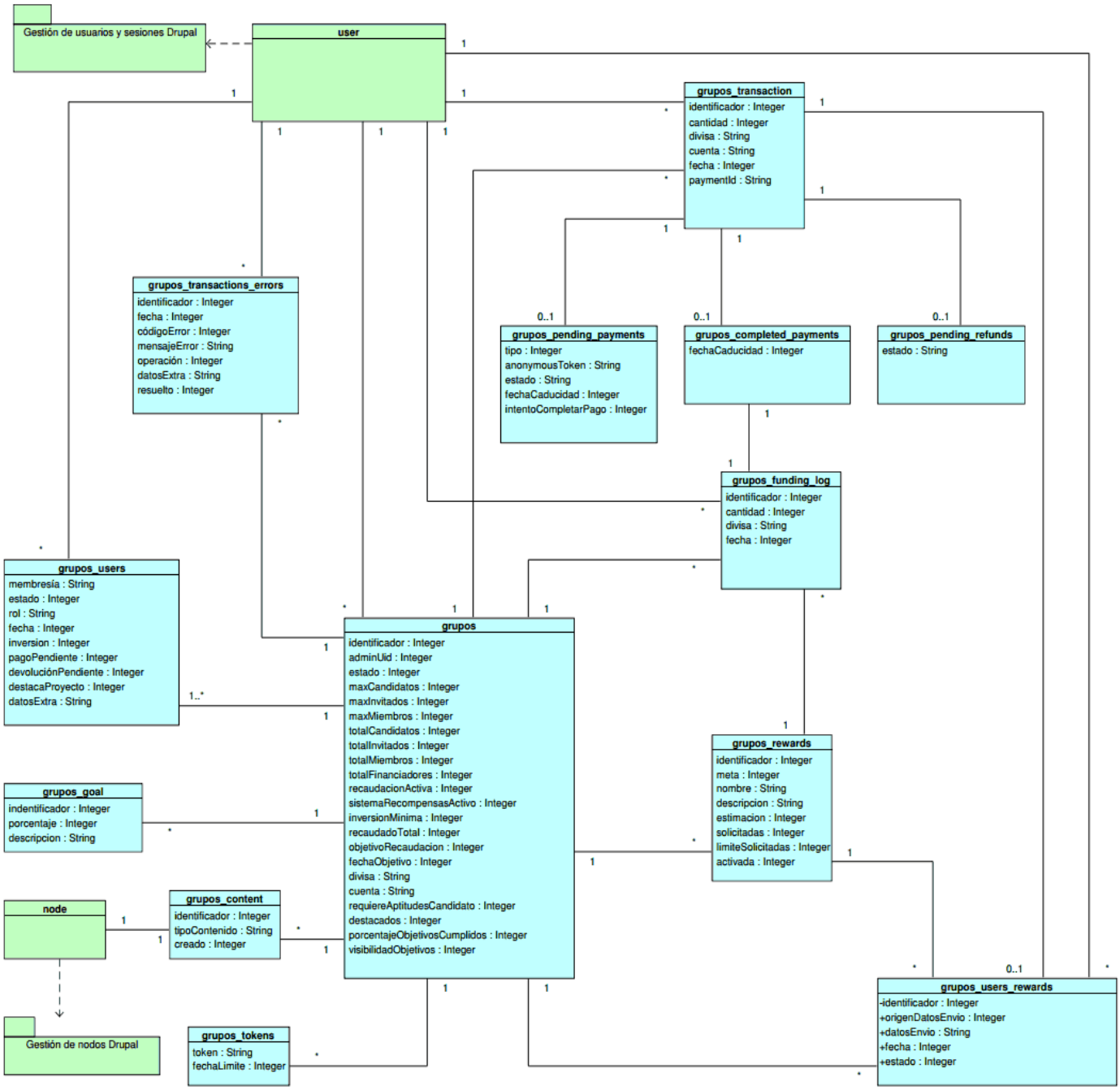
Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el mérito.
- estado– Valor entero que indica si la referencia está pendiente de aprobar o aprobada, por tanto, pública.
- vínculo – Cadena que contiene el tipo de vínculo o relación entre los dos usuarios.
- descripción - Cadena que contiene la descripción de la referencia.

Atributos foráneos.

- uidAutor.user – Valor entero que identifica al usuario que crea la referencia.
- uidReferenciado.user – Valor entero que identifica al usuario que es referenciado.

4.4.2. Estructura base de datos del módulo Grupo



La tabla node pertenece al sistema Drupal y guarda los datos de cada nodo creado, que tal como ya se ha mencionado, cada nodo guarda un elemento de un tipo de contenido ya sea un mensaje de muro, evento o proyecto. Aunque esté aquí representado como una tabla en realidad la componen varías. Una describe los tipos de contenidos disponibles, otra para cada elemento nodo y otra para las versiones de cada nodo, con tal de que sea posible deshacer cambios en el contenido. Además, existen otras tablas donde reside el contenido de cada nodo, que como ya se ha descrito anteriormente, son los llamados campos. Una de ellas define cada campo y sus características. Además para cada campo existe una tabla para el valor del campo y otra para su versión tal como pasa con nodo. Y por último, otra tabla es utilizada para relacionar cada tipo de nodo con los tipos de campo que contiene. Esto da mucha flexibilidad a la hora de crear nuevos tipos de contenidos. A continuación son descritas las tablas con sus atributos y claves primarias.

grupos

Esta tabla registra cada grupo que está asociado a un proyecto en concreto

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica al grupo y que es el mismo que el del nodo proyecto vinculado.
- estado – Valor entero que contiene el estado del grupo. Los valores posibles son(-2 => Eliminado, -1 => Pendiente eliminación, 0 => Cerrado, 1 => Abierto, 2 => Proceso finalización, 3 => Finalizado)
- maxCandidatos – Valor entero que indica el número máximo de candidatos posibles a la vez.
- maxInvitados – Valor entero que indica el número máximo de invitados posibles a la vez.
- maxMiembros – Valor entero que indica el número máximo de miembros posibles a la vez.
- totalCandidatos – Valor entero que indica el número actual de candidatos.
- totalInvitados – Valor entero que indica el número actual de invitados.
- totalMiembros – Valor entero que indica el número actual de miembros.
- totalFinanciadores – Valor entero que indica el número de financiadores distintos del proyecto.
- recaudaciónActiva - Valor entero que indica si el sistema de financiación está activado o no. Los valores posibles son(0 => desactivado, 1 => activado)
- sistemaRecompensasActivo - Valor entero que indica si el sistema de recompensas está activado o no. Los valores posibles son(0 => desactivado, 1 => activado)
- inversiónMínima - Valor entero que indica la inversión mínima inicial obligatoria para un candidato con rol de financiador.
- recaudadoTotal - Valor entero que indica cuanto ha recaudado el proyecto hasta el momento.
- objetivoRecaudación - Valor entero que indica cuanto espera el proyecto recaudar.
- fechaObjetivo - Valor entero que contiene el timestamp de la fecha prevista de finalización del proyecto.
- divisa - Cadena de tres cifras que contiene el símbolo internacional de la divisa utilizada en el proyecto.
- cuenta - Cadena que contiene la cuenta de recaudación vinculada al gestor de transacciones.
- requiereAptitudesCandidato - Valor entero que indica si en la solicitud será requerido una descripción de las aptitudes del candidato.Los valores posibles son (0 => desactivado, 1 => activado)
- destacados - Valor entero que indica el número de usuarios registrados que han destacado el proyecto.
- porcentajeObjetivosCumplidos - Valor entero que contiene el porcentaje de cumplimiento de los objetivos propuestos en el proyecto.
- visibilidadObjetivos - Valor entero que indica para quienes son visibles los objetivos del proyecto. Los valores posibles son(0 => Todos los usuarios, 1 => Solo participantes del grupo, 2 => solo Invitados, Miembros y Administrador, 3 => solo Miembros y Administrador)

Atributos foráneos.

- adminUid.user – Valor entero que identifica al usuario administrador del grupo.

grupos_content

Esta tabla relaciona cada nodo de tipo evento o muro con el grupo.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica la relación.
- tipoContenido – Cadena que contiene el tipo de nodo ya sea evento o muro.
- creado – Valor entero que contiene el timestamp de la fecha en que fue creado el nodo.

Atributos foráneos.

- nid.node – Valor entero que identifica al nodo.

grupos_users

Esta tabla contiene a los usuarios que tienen alguna relación con el proyecto ya sea porque participan como administrador, miembro, invitado o candidato o esté expulsado. Además contiene usuarios que puedan haber destacado el proyecto, invertido o tengan devoluciones pendientes. En definitiva existe algún tipo de información entre el usuario y el proyecto que debe ser registrado.

Clave primaria. (uid.user, gid.grupos)

Atributos.

- membresía – Cadena que contiene el tipo de membresía. Valores posibles son('Administrador', 'Miembro', 'Invitado', 'Candidato', 'NoParticipa')
- estado – Valor entero que contiene el estado del usuario dentro del proyecto. Valores posibles son (-2 => 'Expulsado', -1 => 'Pendiente de expulsión', 1 => 'Pendiente de Aceptación', 2 => 'Aceptado')
- rol – Cadena que contiene el rol del usuario en el proyecto, si es que tiene uno.
- fecha – Valor entero que contiene el timestamp de cuando cuando el usuario cambió su estado.
- inversión - Valor entero que contiene lo invertido por el usuario en el proyecto.
- pagoPendiente – Valor entero que contiene la cantidad del pago pendiente, si es que tiene alguno.
- devoluciónPendiente – Valor entero que contiene la cantidad de la devolución pendiente, si tiene alguna.
- destacaProyecto – Valor entero que indica si ha destacado el proyecto. Valores posibles son(0 => No destacado, 1 => Destacado)
- datosExtra – Cadena que puede contener datos serializados, tales como el resultado de la votación en caso de ser miembro de un proyecto pendiente de finalización.

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario.
- gid.grupos – Valor entero que identifica al grupo.

grupos_goal

Esta tabla gestiona los objetivos añadidos al proyecto.

Clave primaria. (identificador)

Atributos.

- identificador – Valor entero que identifica el objetivo.
- porcentaje – Valor entero que contiene el porcentaje de cumplimiento que representa el objetivo.
- enlace – Cadena que contiene el nombre del mérito.
- descripción - Cadena que contiene la descripción del objetivo.

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.

grupos_rewards

Esta tabla gestiona las recompensas añadidas al proyecto. Y se corresponde en la especificación con la clase Recompensa.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica la recompensa.
- meta – Valor entero que contiene la cantidad mínima para poder solicitar la recompensa.
- nombre – Cadena que contiene el nombre de la recompensa.
- descripción - Cadena que contiene la descripción de la recompensa.
- estimación – Valor entero que contiene en semanas de lo que tardaría en ser recibida la recompensa.
- solicitadas – Valor entero que contiene el número de recompensas solicitadas hasta el momento.
- limiteSolicitadas – Valor entero que contiene la cantidad límite de recompensas que pueden haber solicitadas a la vez. En caso de que el valor sea 0 significa que no hay límite.
- activada – Valor entero que indica si la recompensa está activada o no. Valores posibles son(0 => desactivada, 1 => activada)

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.

grupos_user_rewards

Esta tabla gestiona la relación entre una recompensa y el usuario que la ha solicitado. Y se corresponde en la especificación con la clase RecompensaSolicitada.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica la relación.
- origenDatosEnvío – Valor entero que identifica el origen de los datos. Valores posibles son(0 => datos del perfil del usuario, 1 => datos en el campo datosEnvío)
- datosEnvio – Cadena serializada que contiene los datos de envío.
- fecha – Valor entero que contiene el timestamp de cuando la recompensa solicitada cambió el estado.
- estado - Cadena que contiene la descripción del enlace(opcional)

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.
- uid.user – Valor entero que identifica al usuario.
- rid.grupos_rewards – Valor entero que identifica a la recompensa.
- tid.grupos_transaction – Valor entero que identifica a la transacción.

grupos_transaction

Esta tabla gestiona las transacciones. Aunque en la realidad no se ha llegado a implementar y sus atributos se encuentran duplicados en las tablas grupos_pending_payments, grupos_completed_payments y grupos_pending_refunds. Sin embargo, para facilitar el esquema este ha sido representado de esta forma. Por tanto, tal como está implementado en realidad el atributo tid de la tabla grupos_user_rewards apunta a una de esas tres tablas dependiendo del estado que tenga la recompensa solicitada.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica a la transacción.
- cantidad – Valor entero que contiene la cantidad en unidades monetarias de la transacción.
- divisa – Cadena que contiene la divisa de la transacción.
- cuenta – Cadena que contiene la cuenta de destino de la transacción
- fecha - Valor entero que contiene el timestamp de la fecha de creación de la transacción.
- paymentId – Cadena que contiene el identificador de la transacción en el gestor de transacciones.

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.
- uid.user – Valor entero que identifica al usuario.

grupos_pending_payments

Esta tabla gestiona cada pago pendiente de un usuario hacia un proyecto. Hay ciertos atributos que están fuertemente relacionados con el gestor de transaciones. Como anonymousToken generado como clave para identificar que pago pendiente de un usuario anónimo ha sido completado. El estado, que es el mismo que devuelve el gestor de transacciones. Y otros para funcionamiento interno como el tipo, que sirve a la hora de decidir que hacer con el pago pendiente al desactivar la recaudación o caducar la clave de pago. Por último, por eficiencia el atributo intentoCompletarPago ahorra tener que comprobar el estado de todos los pago pendientes.

Clave primaria. (identificador.grupos_transaction)
Atributos.

- tipo – Valor entero que identifica el tipo de pago pendiente. Útil al renovar el paymentID cuando este caduca. Valores posibles son('0 => no partipante, 1 => participante usual, 2 => participante financiador')
- anonymousToken – Cadena generada que identifica el pago pendiente de un usuario anónimo.
- estado – Cadena que contiene el estado del pago pendiente retornado por el gestor de transacciones.
- fechaCaducidad - Valor entero que contiene el timestamp de cuando caducará el paymentId.
- intentoCompletarPago – Valor entero que indica si el usuario ya ha intentado completar el pago. Util al realizar las comprobaciones automáticas del estado de los pagos pendientes. Valores posibles son

(0 => aun no ha intentado completarlo, 1 => lo ha intentado)
Atributos foráneos.

- identificador.grupos_transaction – Valor entero que identifica la transacción.

grupos_completed_payments

Esta tabla gestiona los pagos completados de los usuarios hacia los proyectos. Cada fila de esta tabla existirá como máximo por el tiempo establecido para realizar devoluciones del gestor de transacciones.

Clave primaria. (identificador.grupos_transaction)
Atributos.

- fechaCaducidad - Valor entero que contiene el timestamp de cuando caducará el periodo para realizar una posible devolución del pago completado.

Atributos foráneos.

- identificador.grupos_transaction – Valor entero que identifica la transacción.

grupos_pending_refunds

Esta tabla gestiona las devoluciones pendientes de completar.

Clave primaria. (identificador.grupos_transaction)
Atributos.

- estado – Cadena que contiene el estado de la devolución retornado por el gestor de transacciones.

Atributos foráneos.

- identificador.grupos_transaction – Valor entero que identifica la transacción.

grupos_transactions_errors

Esta tabla gestiona los errores en las transacciones realizadas con el gestor de transacciones. Debido a que es un sistema ajeno a la plataforma, pueden producirse errores no previstos, por tanto, han de ser registrados para tomar las medidas necesarias. Solo se tienen en cuenta los errores a la hora de completar el pago o realizar devoluciones.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el error.
- fecha – Valor entero que contiene el timestamp de la fecha en que se produjo el error.
- códigoError – Valor entero que contiene el código de error retornado por el gestor de transacciones.
- mensajeError – Cadena que contiene el mensaje de error retornado.
- operación - Valor entero que identifica el tipo de operación. Valores posibles son (0 => Devolución, 1 => Pago')
- datosExtra – Cadena serializada con información extra necesaria para resolver el error.
- resuelto – Valor entero que indica si ha sido o no resuelto. Valores posibles son (0 => No resuelto, 1 => Resuelto)

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.
- uid.user – Valor entero que identifica al usuario.

grupos_funding_log

Esta tabla mantiene un registro de los pagos completados realizados a los proyectos. A diferencia de la tabla grupos_completed_payments esta no es borrada pasada un tiempo. Solo en el caso de realizarse devoluciones tanto en la tabla grupos_completed_payments y esta son borradas las filas que correspondan.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica la financiación hacia los proyectos.
- cantidad – Valor entero que contiene la cantidad en unidades monetarias del pago.
- divisa – Cadena que contiene la divisa del pago.
- fecha – Valor entero que contiene el timestamp de la fecha en que se financió.

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.
- uid.user – Valor entero que identifica al usuario.
- rid.grupos_rewards – Valor entero que identifica a la recompensa.
- tid.grupos_transaction – Valor entero que identifica a la transacción.

grupos_tokens

Esta tabla gestiona los tokens utilizados a la hora de validar una cuenta introducida por el administrador de un proyecto para la recaudación. El token es enviado por email a la cuenta seleccionada en un enlace. Cuando el usuario va al enlace, el sistema identifica que la cuenta es válida.

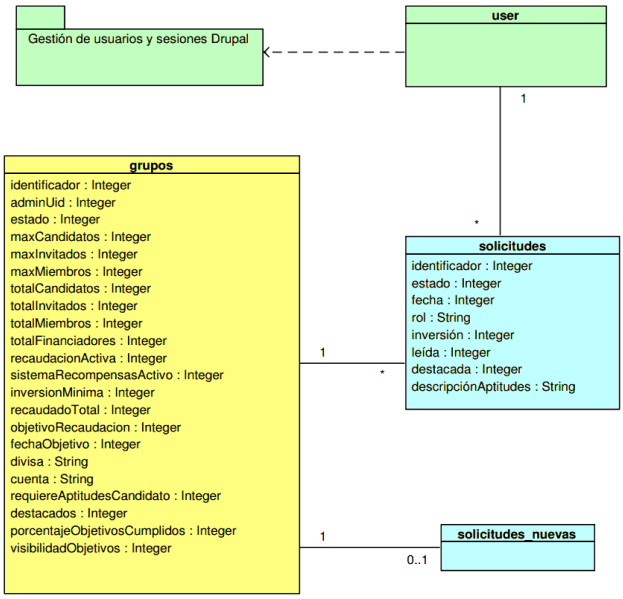
Clave primaria. (gid.grupos)
Atributos.

- token – Cadena que contiene el token generado para la validación.
- fechaLimite – Valor entero que contiene el timestamp de la fecha en que caduca el token. Será renovado y vuelto a enviar el nuevo a la cuenta indicada por el usuario.

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.

4.4.3. Estructura base de datos del módulo Solicitud



A continuación son descritas las tablas con sus atributos y claves primarias.

solicitudes

Esta tabla gestiona las solicitudes enviadas a los proyectos para poder participar.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica la financiación hacia los proyectos.
- estado – Valor entero que indica el estado de la solicitud. Valores posibles son(-1 => Rechazada, 0 => Pendiente, 1 => Aprobada)
- fecha – Valor entero que contiene el timestamp de la fecha en que cambió de estado.
- rol – Cadena que contiene el rol elegido por el candidato que desempeñará en el proyecto.
- inversión – Valor entero que contiene la financiación inicial prometida por el usuario.
- leída – Valor entero que indica si el administrador del proyecto ha leído la solicitud. Valores posibles son(0 => no leída, 1 => leída)
- destacada – Valor entero que indica si el administrador del proyecto ha destacado la solicitud. Valores posibles son(0 => no destacada, 1 => destacada)
- descripciónAptitudes – Cadena con la descripción de aptitudes introducida por el candidato.

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.
- uid.user – Valor entero que identifica al usuario.

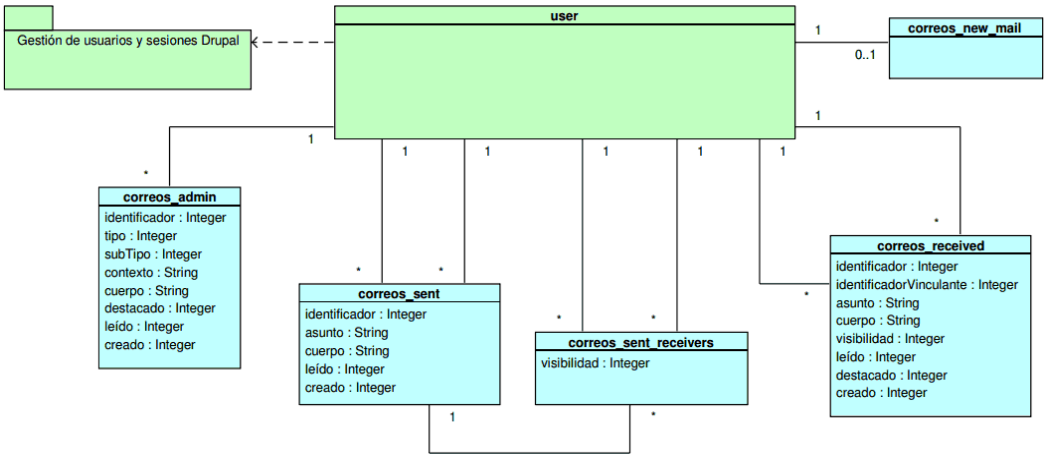
solicitudes_nuevas

Esta tabla indica si el proyecto ha recibido solicitudes nuevas. En caso de existir una entrada de un proyecto en la tabla, es que ha recibido solicitudes nuevas.

Clave primaria. (gid.grupos)
Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.

4.4.4. Estructura base de datos del módulo Correo



A continuación son descritas las tablas con sus atributos y claves primarias.

correos_sent

Esta tabla gestiona los correos enviados por un usuario.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el correo enviado.
- asunto – Cadena que contiene el asunto del correo.
- cuerpo – Cadena que contiene el cuerpo del correo.
- leído – Valor entero que indica si ha leído el correo. Valores posibles son(0 => no leído, 1 => leído)
- creado - Valor entero que contiene el timestamp de la fecha de cuando el correo fue enviado.

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario que crea el correo.

correos_sent_receivers

Esta tabla gestiona los destinatarios de un correo enviado. En caso

Clave primaria. (identificador.correos_sent, uidReceiver.user)
Atributos.

- visibilidad – Valor entero que indica si el usuario es visible u oculto para el resto de destinatarios.

Atributos foráneos.

- Identificador.correos_sent – Valor entero que identifica el correo enviado.
- uidSender.user – Valor entero que identifica al usuario que envió el correo.
- uidReceiver.user – Valor entero que identifica al usuario destinatario del correo.

correos_received

Esta tabla gestiona los correos recibidos por los usuarios.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el correo recibido.
- identificadorVinculante – Valor entero que vincula un mismo correo recibido por varios usuarios.
- asunto – Cadena que contiene el asunto del correo.
- cuerpo – Cadena que contiene el cuerpo del correo.
- visibilidad – Valor entero que indica si el usuario es visible u oculto para el resto de destinatarios.
- leído – Valor entero que indica si ha leído el correo. Valores posibles son(0 => no leído, 1 => leído)
- destacado – Valor entero que indica si el correo está destacado. Valores posibles son
(0 => no destacado, 1 => destacado)
- creado - Valor entero que contiene el timestamp de la fecha de cuando el correo fue enviado.

Atributos foráneos.

- uidSender.user – Valor entero que identifica al usuario que envió el correo.
- uidReceiver.user – Valor entero que identifica al usuario destinatario del correo.

correos_new_mail

Esta tabla indica si el usuario ha recibido correos nuevos. En caso de existir una entrada de un usuario en la tabla, es que ha recibido correos nuevos.

Clave primaria. (uid.user)
Atributos foráneos.

- uid.user – Valor entero que identifica al usuario.

correos_admin

Esta tabla gestiona los correos que recibe el administrador del sistema ya sea desde el formulario de contacto como al denunciar elementos o usuarios de la plataforma.

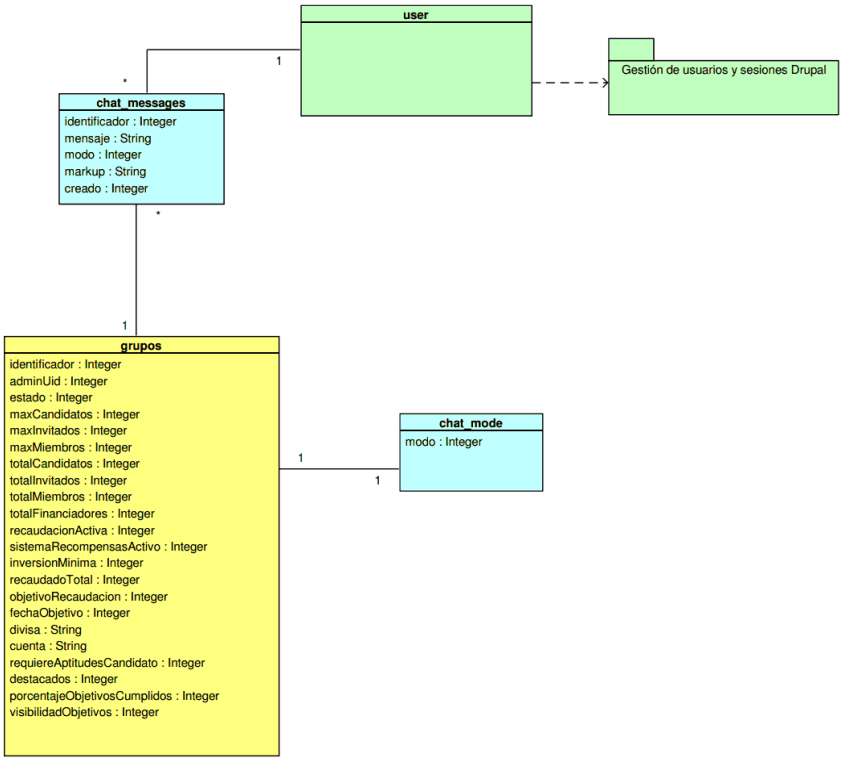
Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el correo recibido.
- tipo – Valor entero que identifica el tipo de correo. Valores posibles son(0 => Duda, 1 => Problema técnico, 2 => Reclamación, 3 => Mejoras, 4 => Denúncia)
- subTipo – Valor entero que identifica el subtipo de correo. Útil para las denuncias.
- contexto – Cadena que contiene el contexto, normalmente el identificador del elemento o usuario.
- cuerpo – Cadena que contiene el cuerpo del correo.
- leído – Valor entero que indica si ha leído el correo. Valores posibles son(0 => no leído, 1 => leído)
- destacado – Valor entero que indica si el correo está destacado. Valores posibles son
(0 => no destacado, 1 => destacado)
- creado - Valor entero que contiene el timestamp de la fecha de cuando el correo fue enviado.

Atributos foráneos.

- uid.user – Valor entero que identifica al usuario que envió el correo.

4.4.5. Estructura base de datos del módulo Chat



A continuación son descritas las tablas con sus atributos y claves primarias.

chat_messages

Esta tabla gestiona los mensajes de chat de los proyectos.

Clave primaria. (identificador)
Atributos.

- identificador – Valor entero que identifica el mensaje de chat.
- mensaje – Cadena que contiene el mensaje.
- modo – Valor entero que contiene la visibilidad del mensaje, si es privado solo para administrador y miembros o público para invitados también. Valores posibles son('0 => Privado, 1 => Público')
- markup – Código html generado del mensaje junto con la fecha y el usuario. Para mayor eficiencia.
- creado - Valor entero que contiene el timestamp de la fecha de cuando el mensaje fue enviado.

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.
- uid.user – Valor entero que identifica al usuario.

chat_mode

Esta tabla registra el modo de visibilidad del chat para cada proyecto.

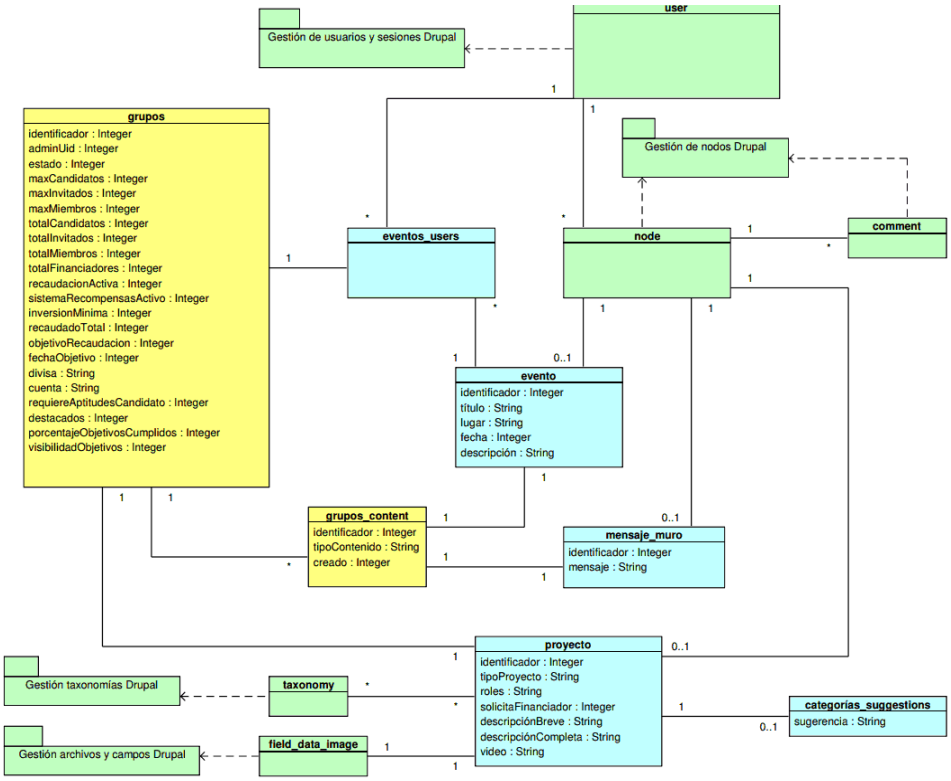
Clave primaria. (gid.grupos)
Atributos.

- modo – Valor entero que contiene la visibilidad del chat, si es privado solo para administrador y miembros o público para invitados también. Valores posibles son('0 => Privado, 1 => Público')

Atributos foráneos.

- gid.grupos – Valor entero que identifica al grupo.

4.4.6. Estructura base de datos de Proyecto, Evento y Muro



La tabla comment contiene los comentarios asociados a cada nodo, los cuáles puede elegirse que sean o no activados. La tabla taxonomy en realidad está compuesta por varias tablas y gestiona las taxonomías y vocabulario de sitio. Para esta plataforma se han utilizado las taxonomías para las categorías del proyecto. También se ha utilizado la tabla del campo imagen para guardar la información sobre la imagen asociada al proyecto y que es posible ver en la página principal del proyecto. A continuación son descritas las tablas con sus atributos y claves primarias.

proyecto

Esta tabla por claridad se ha representado en una sola caja pero realmente el nodo proyecto y sus campos están implementados en varías tablas, al menos una por cada campo.

Clave primaria. (nid.node)
Atributos.

- tipoProyecto – Cadena que identifica el tipo de proyecto. Valores posibles 'Emprendedor' o 'Social'
- roles – Cadena que contiene los roles o actividades requeridos, separados por una coma.
- solicitaFinanciador – Valor entero que indica si requiere financiador.
- descripciónBreve – Cadena que contiene la descripción breve del proyecto.
- descripciónCompleta – Cadena que contiene la descripción completa del proyecto.
- vídeo – Cadena que contiene el enlace web al vídeo del proyecto.

Atributos foráneos.

- nid.node – Valor entero que identifica el nodo.
- tid.taxonomy – Valor entero que identifica la categoría del proyecto.
- fid.field_data_image – Valor entero que identifica la imagen del proyecto.

categorías_suggestions

Esta tabla gestiona las categorías sugeridas por cada administrador que no haya encontrado una categoría de las que hay por defecto que se ajuste a su proyecto para posterior revisión.

Clave primaria. (nid.node)
Atributos.

- sugerencia – Cadena que contiene la categoría sugerida.

Atributos foráneos.

- nid.node – Valor entero que identifica el nodo.

evento

Esta tabla por claridad se ha representado en una sola caja pero realmente el nodo evento y sus campos están implementados en varías tablas, al menos una por cada campo.

Clave primaria. (nid.node)
Atributos.

- título – Cadena que contiene el título del evento.
- lugar – Cadena que contiene el lugar donde se producirá el evento.
- fecha – Valor entero que contiene el timestamp de la fecha de cuando se producirá el evento.
- descripción – Cadena que contiene la descripción del evento.

Atributos foráneos.

- nid.node – Valor entero que identifica el nodo.

eventos_users

Esta tabla gestiona los usuarios que han decidido participar en un evento de un proyecto. Por tanto, solo aparecen entradas para usuarios participantes de estos.
Clave primaria. (identificador.evento, gid.grupos, uid.user)
Atributos foráneos.

- identificador.evento – Valor entero que identifica el nodo evento.
- gid.grupos – Valor entero que identifica el grupo.
- uid.user – Valor entero que identifica el usuario.

mensaje_muro

Esta tabla por claridad se ha representado en una sola caja pero realmente el nodo mensaje de muro y sus campos están implementados en varías tablas, al menos una por cada campo.

Clave primaria. (nid.node)
Atributos.

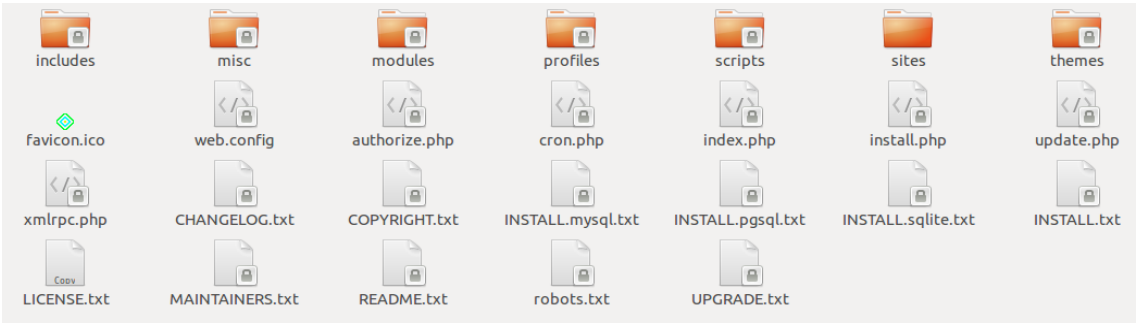
- mensaje – Cadena que contiene el mensaje del mensaje de muro.

Atributos foráneos.

- nid.node – Valor entero que identifica el nodo.

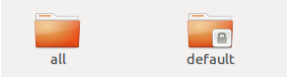
4.5. Estructura de los directorios y archivos

En esta sección presentamos el listado de directorios y archivos más importantes para el funcionamiento de Drupal y de la plataforma desarrollada. Tal como ya se mencionó anteriormente, Drupal hace una separación entre su núcleo y las extensiones y nuevos temas que sean instalados.

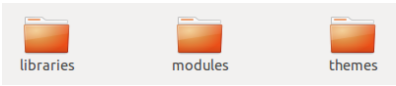


Esta imagen corresponde al directorio principal de Drupal. Como se puede apreciar todos los directorios y archivos corresponden al núcleo, teniendo en archivos con funciones comunes, en misc recursos comunes, tales como imágenes o código en javascript. En módulo encontramos el grueso del núcleo, con todos los módulos que incorporados en la instalación inicial. En la carpeta scripts podemos encontrar código para el mantenimiento de la web además de ejemplos sobre como configurar los demonios(trozos de código que realizan tareas en segundo plano automáticamente). El directorio themes aloja los temas por defecto. El resto de archivos son utilizados para la ejecución de cada petición web, que por defecto es install.php, actualización del sistema, entre otras funciones.

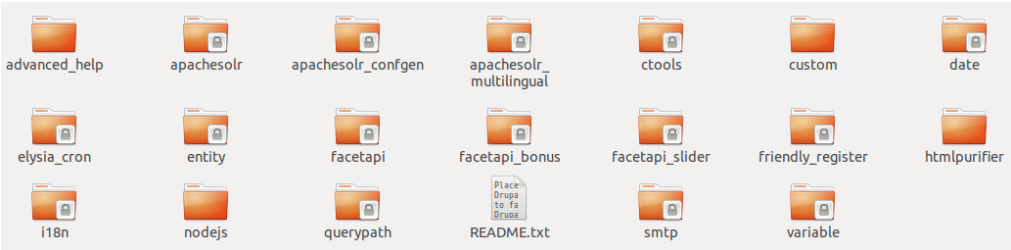
Nuestra plataforma se encuentra en la carpeta sites, donde residen los nuevos módulos y temas instalados después de la instalación inicial.



Adentrándonos en este directorio encontramos dos directorios, default, donde se encuentran la configuración global, los datos de conexión con la base de datos y archivos subidos por los usuarios, tales como imagenes. El segundo directorio, all, es donde está alojado el sitio web.



Dentro encontramos el directorio libraries que para esta plataforma es utilizado para albergar el wysiwyg tinymce, que es código en javascript añadido a la web que extiende las funcionalidades de un área de texto de formulario web para que funcione como un editor web en línea. El directorio themes alberga nuevos temas instalados, siendo utilizado para esta plataforma el tema llamado Basic, el cual nos ha permitido modificarlo a nuestra medida. Y por último modules que alberga todos los módulos instalados y creados.



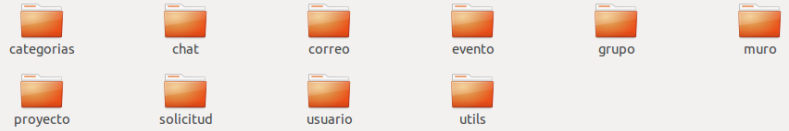
A grandes rasgos los módulos instalados más importantes son elysia cron que gestiona los cron jobs, un demonio que acciona tareas periódicamente de mantenimiento en la plataforma. Especificamente, han sido creados varios cron jobs, algunos de ellos se encargan de borrar datos demasiado antiguos. Existe otro relacionado con los pagos pendientes que renuevan las claves de pago y otro que comprueba si los pagos o devoluciones han sido completados. También existe uno que desbloquea usuarios que han sido expulsados de proyectos despues de 4 meses.

Un módulo también esencial es apachesolr, que se encarga de indexar y buscar contenido. Este módulo además tiene configurado un cron job para la indexación periódica de nuevo contenido.

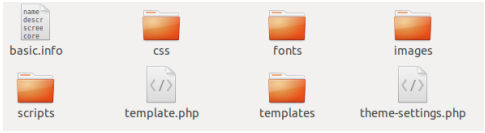
Otro módulo no menos importante es nodejs, el cual gestiona las actualizaciones del contenido web cuando hay cambios en el servidor y por tanto, permite que en vez de que sea el cliente quien de forma periódica pregunte al servidor si hay cambios, es el propio servidor quien avisa al navegador del cliente. Entre otros aspectos proporciona immediatez al chat, al muro, correo y eventos en sus actualizaciones.

Además hay otros módulos como smtp el cual proporciona la posibilidad de enviar correos electrónicos fuera de la plataforma. También friendly_register encargado de que en el momento de registrarse indicar si un email y nombre de usuario están disponibles o date que proporciona poder seleccionar la fecha mediante popups, ventanas emergentes. Y por último htmlpurifier, un potente filtrado de contenido para la seguridad de la plataforma.

El resto son módulos necesarios para el funcionamiento de los otros.



Y en la carpeta custom encontramos los módulos propiamente creados para la plataforma. En los cuales se encuentran los archivos ya mencionados con extensiones .install, .module y .inc además de archivos en javascript utilizados para mejorar la experiencia del usuario y minimizar el número y tamaño de las peticiones al servidor.



Volviendo hacia atrás a la carpeta all, y entrando en themes/basic, encontramos los archivos y directorios del tema Basic, encargados de configurar el aspecto de la plataforma.

4.6. Entorno de funcionamiento de la plataforma

Para el desarrollo y puesta en marcha de la plataforma ha sido utilizado un servidor compartido con una partición propia de Linux, exactamente la distribución CentOS. El servidor elegido fue seleccionado con la característica principal de poder tener acceso de superusuario a la partición de CentOS asignada para poder instalar aplicaciones necesarias para la plataforma. Las tecnologías utilizadas son las siguientes.

- Sistema operativo CentOS
- Servidor web Apache 2.4.7
- Base de datos MySQL 5.5.37
- PHP 5.4.25
- Drupal 7.28
- Tomcat 7
- ApacheSolr
- NodeJS

La utilización Tomcat es un requisito para el indexador de contenido ApacheSolr. Este servlet es utilizado para poder realizar búsquedas de proyectos a través de la plataforma, dando la opción instalando ciertos módulos de drupal además del que sirve para conectar este último con ApacheSolr.

NodeJS es utilizado para las actualizaciones activadas desde el servidor, es necesario no solo tener NodeJs instalado en el servidor sino además el módulo Drupal que realiza al conexión entre ambos.

Para la implementación ha sido utilizado NetBeans, con el complemente Xdebug para la depuración de código.

Los requisitos mínimos para el correcto funcionamiento de Drupal 7.28 son.

- Sistema operativo Windows, Linux
- Servidor web Apache 1.3, Nginx 0.7, Hiawatha, Microsoft IIS
- Base de datos MySQL 5.0.15, PostgreSQL 8.3, SQLite 3.3.7
- PHP 5.2.5
- 15 Mb de espacio en disco

El dominio escogido para la plataforma es http://project-harvester.es.

5. Diseño de la interfaz

A continuación presentamos un esbozo del interfaz de usuarios para la página principal y la página de proyecto.

5.1. Página principal

Cabecera

En la cabecera además del logo del sitio web encontramos información del usuario que ha iniciado sesión. En la Figura anterior en la posición 1 encontramos el nombre del usuario registrado que ha iniciado sesión, en caso de no haber iniciado aun sesión encontraríamos el botón para el inicio de sesión.
En la posición 2 está ubicado el botón para cerrar sesión en caso de que el usuario registrado haya iniciado sesión, o en caso de no haber iniciado sesión encontraríamos el botón para registrarse.
La cabecera es una parte fija de la plataforma por tanto, aparece en todas las pantallas de cualquiera de las funcionalidades disponibles, de esta forma el usuario está en todo momento informado de que sesión tiene abierta y la opción de cerrarla de forma directa.

Menú principal

El menú principal de la Figura anterior ubicado en la posición 3 encontramos diferentes opciones para un acceso directo por parte del usuario. Aunque todas las opciones mostradas son autoexplitivas, explicaremos brevemente un par de ellas, Usuarios, Aportaciones.

En la pantalla de usuarios, aparece en el area central el listado de usuarios bloqueados por el usuario registrado y el listado de los destacados por este. Estos dos listados son utilizados para quitar o mover de un listado al otro los usuarios, por tanto, gestiona los usuarios marcados, tanto como destacados como bloqueados por el usuario registrado.

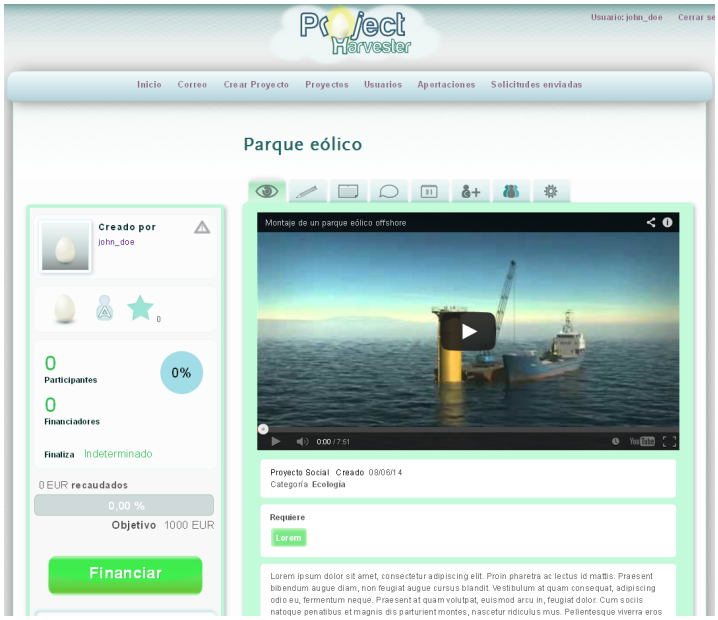
En la página de aportaciones, aparece en el area central el listado de proyectos que el usuario ha financiado, junto con la información de cuanto ha financiado, cuando hizo el pago, además de la recompensa, si esta fue solicitada. También es mostrada una pestaña para poder ver las recompensas, tanto las que tiene pendientes de pago, como las completadas, las que están pendientes de envío y las pendientes de devolución de pago, para así poder gestionarlas o consultar datos relevantes para el usuario sobre estas.E l menú principal es una parte fija de la plataforma por tanto, aparece en todas las pantallas de cualquiera de las funcionalidades disponibles, de esta forma proveemos un acceso directo a las más importantes.

Área central

En el área central encontramos diferentes elementos el primero de ellos, en la posición 4, son botones para elegir el tipo de proyecto a visualizar. El primero para emprendedores, el segundo para todos, y el tercero para proyectos de carácter social. En la posición 5, encontramos el buscador de proyectos, una vez introducidos los datos aparece en el lado izquierdo del area central un listado de filtros los cuales dependen de los proyectos encontrados en la búsqueda realizada.

Por último en la posición 6 encontramos el listado de proyectos, los cuáles como ya ha sido mencionado anteriormene para la página principal, el título, imagen del proyecto, tipo de proyecto, estado del mismo, categoría, descripción breve y además información tanto de la finanación del proyecto como del progreso.

5.2. Página del proyecto



Área central

El área central está dividida en dos partes. La primera parte la podemos encontrar a la izquierda y es una parte fija para todas las páginas relacionadas con el proyecto. Esta sección muestra información del proyecto, tanto el autor, como la membresía, estado y rol del usuario que visita la página con respecto al proyecto entre otros elementos. Más abajo muestra información más general sobre el proyecto además de la financiación. Por último, en la parte inferior de esta sección aparecen los botones con opciones de asociación al grupo que dependen de la membresía y estado del usuario con respecto al proyecto. Estos botones proveen las funcionalidades de enviar solicitud, salir, eliminar y finalizar proyecto.

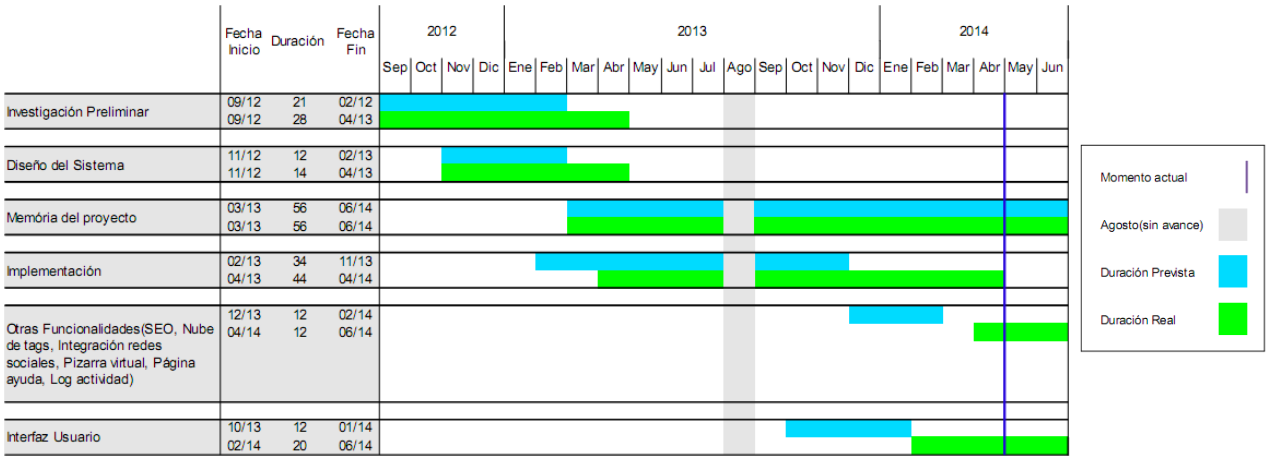
La segunda parte del área central que se encuentra a la derecha, es el proyecto en sí, existen un número determinado de pestañas sobre este para llevarnos a las diferentes páginas relacionadas con el proyecto. Estas páginas son las de información completa del proyecto, edición, chat, muro, eventos, solicitudes recibidas, usuarios del proyecto y configuración.

Por tanto, proveemos al usuario con toda la información esencial del proyecto, estando dentro de él, además de acceso directo a las funcionalidades del mismo.

6. Meta gestión

En esta sección exponemos la planificación y gestión del proyecto, tiempo empleado y coste estimados.

6.1. Planificación, tiempo previsto y real



La duración del diagrama de gantt anterior representa la duración en semanas. Han sido juntadas tanto la duración prevista como la duración real en un mismo diagrama para apreciar mejor la desviación.

Cada etapa está compuesta por una serie de tareas que serán descritas a continuación.

La etapa de investigación preliminar está compuesta primeramente por la definición preliminar del proyecto, es decir, un esbozo de este realizando una lluvia de ideas y organizándolas de forma coherente. De esta forma delimitar las funcionalidades más básicas a grandes rasgos que tendría la plataforma.

Una vez determinado que el medio para utilizar esta plataforma sería la web, fue realizado un análisis y comparativa entre diversos CMS para escoger el que sería utilizado para desarrollar el proyecto. En este caso, fue Drupal, del cual no disponíamos de conocimiento previo, por tanto, hubo que planificar el tiempo necesario para aprender a utilizarlo correctamente.

Para profundizar aun más en las funcionalidades y características y poder lograr una diferencia real en relación con otras plataformas similares, fue realizado un análisis y comparativa entre estas. Y con esto acabó la etapa de investigación preliminar.

La siguiente etapa es diseño del sistema, donde realizamos una definición detallada de los objetivos y restricciones. Fueron especificadas las clases necesarias, los casos de uso y procedimientos, además de la base de datos y un esbozo de la interfaz de usuario.

Además se realizó un análisis más detallado del CMS escogido para aprovechar al máximo la potencia de este y poder entender como dar los siguientes pasos de forma precisa en la implementación. Por último, en esta etapa se realizó un análisis de servidores que cumplían los requerimientos iniciales, para poder empezar a trabajar con ellos en la siguiente etapa.

En la etapa de implementación, fueron creados los módulos necesarios, para extender las funcionalidades que Drupal nos brinda inicialmente y que se ajusten a los requerimientos de nuestro proyecto. El método para realizar el desarrollo utilizado fue un método ágil parecido a SCRUM, que consiste en implementar un pequeño número de funcionalidades en un corto periodo de tiempo, probarlas y modificar lo necesario hasta obtener los resultados esperados. Una vez obtenidos esos resultados, es realizado el mismo proceso con otra remesa de funcionalidades.

Entretanto, la memoria del proyecto fue siendo redactada conforme progresaba el desarrollo de la plataforma.

Una de las últimas etapas fue la de interfaz de usuario, en la que ha sido adecuada la plataforma para hacerla lo más amigable y fácil de usar para todos los usuarios que potencialmente la utilicen.

Por último, ha habido una etapa de más para implementar funcionalidades que ayuden a mejorar la plataforma y su visibilidad en Internet.

Las desviaciones entre lo previsto y lo real han sido principalmente a raíz de la carencia de conocimiento previo de Drupal y sobretodo del desarrollo web. Estas causas provocaron que la estimación fuese bastante más pequeña de lo que debía haber sido, empezando a desviarse ya en la tarea de aprender a utilizar Drupal. Esto condicionó también que en la etapa de implementación hubiesen desviaciones importantes, que fueron disminuyendo conforme pasó el tiempo y el conocimiento adquirido sobre Drupal fue mucho mayor, pero no fue suficiente para equilibrar los tiempos previstos y reales.

6.2. Costes previstos y reales

Cada una de las semanas de las figuras siguientes están compuestas por 20 horas laborables. Han sido calculadas de esta forma teniendo en cuenta la normativa de PFC de la FIB para un proyecto de ingenería técnica. Según esta normativa el proyecto representa 22,5 crédito, y cada crédito equivale a 20 horas. Estos 22,5 créditos han sido contabilizados por cuatrimestre. Por tanto, tal como muestra las figuras la duración prevista son 1000 horas pero la real ha sido de 1500 horas. Sin embargo, por claridad también están incluidas las semanas.

Ha sido considerado que para realizar el proyecto son necesarios un analista, un diseñador y un desarrollador. Para cada uno de estos recursos se ha estimado un precio por hora que servirá para calcular, según las horas previstas y reales, los costes.

Para calcular los siguientes precios, no ha sido posible encontrar un cifra concreta para cada uno ya que depende de muchos factores, tales como titulación, experiencia, reputación, etc. Pero ha sido realizada en base a una media de la información encontrada.

- Analista 40€ por hora.
- Diseñador 35€ por hora.
- Desarrollador 25€ por hora.

Cálculo previsto

	Recurso	Cantidad	Semanas	Horas	Precio por hora	Coste total
Investigación Preliminar	Analista	1	21	200	40	8000
Diseño del Sistema	Analista	1	12	120	40	4800
	Diseñador	1	12	120	35	4200
Memória del proyecto	Analista	1	56	40	40	1600
	Desarrollador	1	56	40	25	1000
Implementación	Desarrollador	1	34	240	25	6000
Otras Funcionalidades(SEO, Nube de tags, Integración redes sociales, Pizarra virtual, Página ayuda, Log actividad)	Desarrollador	1	12	40	25	1000
Interfaz Usuario	Diseñador	1	12	100	35	3500
	Desarrollador	1	12	100	25	2500
				1000		32600

Cálculo real

	Recurso	Cantidad	Semanas	Horas	Precio por hora	Coste total
Investigación Preliminar	Analista	1	28	300	40	12000
Diseño del Sistema	Analista	1	14	160	40	6400
	Diseñador	1	14	160	35	5600
Memória del proyecto	Analista	1	56	40	40	1600
	Desarrollador	1	56	40	25	1000
Implementación	Desarrollador	1	44	380	25	9500
Otras Funcionalidades(SEO, Nube de tags, Integración redes sociales, Pizarra virtual, Página ayuda, Log actividad)	Desarrollador	1	12	40	25	1000
Interfaz Usuario	Diseñador	1	20	200	35	7000
	Desarrollador	1	20	200	25	5000
				1520		49100

7. Conclusiones

Una vez concluida la fase de desarrollo prevista para este proyecto, haber hecho las mejoras necesarias y solucionado los fallos pueden extraerse las siguientes conclusiones.

Después de analizar la plataforma desarrollada http://project-harvester.es podemos concluir que los objetivos del proyecto han sido cumplidos: Actualmente disponemos de una versión Alfa de la plataforma que permite publicar proyectos con la información necesaria para que las personas que puedan desempeñar ciertos roles o actividades dentro del proyecto puedan solicitar participar en éste. Además la plataforma facilita la comunicación entre los usuarios a través de diferentes vías de forma rápida y dispone de un sistema de financiación con recompensas muy completo. En realidad, consideramos que la plataforma se encuentra a medio camino entre una versión alfa y una versión beta, ya que al tener una estructura modular compleja y, como hemos explicado en la Sección 6, al haber utilizado un método ágil de desarrollo (al estilo del método SCRUM) consideramos que los módulos han pasado una fase de verificación y prueba.

Hay ciertas funcionalidades menores, planificadas que no han podido aún ser implementadas, tales como el log de actividades, la traducción a distintos idiomas o la pizarra virtual que han sido dejados como futura extensiones.

Respecto a las conclusiones técnicas, el desarrollo de la plataforma sobre Drupal inicialmente fue laborioso y lento debido a la carencia de experiencia previa no sólo en CMS como Drupal sino en el desarrollo web en general. Aunque pasado el tiempo y habiendo ganado destreza con este gestor de contenidos, considero acertada la elección del CMS Drupal ya que pudimos comprobar que efectivamente ofrece muchas herramientas que facilitan la integración de código propio en PHP y la adaptación de la base de datos a nuestras necesidades. Aunque si hay que concluir que las partes más complicadas sobre todo tenían origen en módulos ajenos instalados tales como los que hacen de puente entre Drupal y NodeJS o ApacheSolr, ya que estos obligaban a revisar el código de sus módulos concienzudamente para aprovechar las características que podían ofrecer.

La plataforma aun tiene un largo camino para mejorar y extender la experiencia del usuario y las funcionalidades que puede ofrecer, tales extensiones son descritas en la siguiente sección.

Finalmente, las expectativas en relación a las capacidades adquiridas en ingeniería informática de gestión en la FIB se han cumplido por completo: todos los obstáculos técnicos se han superado sin inconveniente.

8. Futuras extensiones

Una de las actividades pendientes es la relacionada con el análisis de la usabilidad de la plataforma. Tenemos previsto mejorar y aumentar la experiencia del usuario y las herramientas para lograr los objetivos de esta plataforma son variadas.
Un ejemplo es poder analizar la experiencia y formación del usuario de su perfil o los proyectos en los que participa o ha solicitado participar para determinar que proyectos priorizar a la hora de mostrarlos en la página principal. De esta forma los proyectos mostrados son personalizados según el usuario y esto puede aumentar en gran medida las posibilidades de encontrar el más adecuado.

Además, realizando la operación inversa el administrador del grupo podría averiguar fácilmente que usuarios podrían ser los más adecuados para participar en los proyectos y así enviar invitaciones a estos a participar.

Para aumentar la información sobre novedades en los proyectos en que participa sería interesante que dispusieran de un área de notificaciones. Además, el administrador y los miembros podrían tener acceso a un log de actividad y de esta forma tener más información de lo que ocurre dentro del proyecto.

Con tal de reducir la carga de trabajo del administrador podría asignar permisos tales como aceptar o rechazar solicitudes a otros miembros.

Mejorando las funcionalidades de los eventos podrían añadirse eventos personales o lo que es lo mismo tareas para ser asignadas a los miembros con una fecha de inicio y fin previstas.

Aunque ya existen plataformas de repositorios de ideas, podría ser una buena característica tener uno propio además mantener enlaces a otros repositorios para difundir la innovación.

En el área de comunicación podría ser incorporado un sistema de videollamada entre miembros y administrador. Además, la creación y edición de documentos de forma compartida en cada proyecto, tal como google drive proporcionaría una funcionalidad de gran utilidad.

Con tal de internacionalizar la plataforma sería útil incorporar la traducción a distintos idiomas. Además la asignación de los proyectos por zonas, acompañadas por un potente sistema de búsqueda por zona facilitaría aun más la internacionalización.

Realización de pruebas funcionalidades y de usabilidad con usuarios reales a fin de llevar la plataforma a una fase Beta y luego a una fase de producción.

Otra característica que incentivaría en gran medida y mejoraría el sistema de recompensas, es que puedan haber patrocinadores de ciertas recompensas, de forma que estos mismos creen recompensas a cambio de aparecer anunciados en el proyecto.

9. Referencias

1. **Wikipedia.** Glosario de términos http://es.wikipedia.org

2. **Dolors Costal, Xavier Franch, M. Ribera Sancho, Ernest Teniente.** Enginyeria del software Especificació

3. **PHP manual.** http://www.php.net/manual/es/

4. **W3C schools.** http://www.w3schools.com/

5. **JQuery documentation.** http://api.jquery.com/

6. **NodeJS.** http://nodejs.org/

7. **Drupal.** https://www.drupal.org/

8. **Netbeans.** https://netbeans.org/

9. **Xdebug.** http://xdebug.org/

10. **Apache Tomcat.** http://tomcat.apache.org/

11. **ApacheSolr.** http://lucene.apache.org/solr/

Apéndice

Índice

Apéndice I - Diagramas de clases completos

Diagrama de clases de Usuario

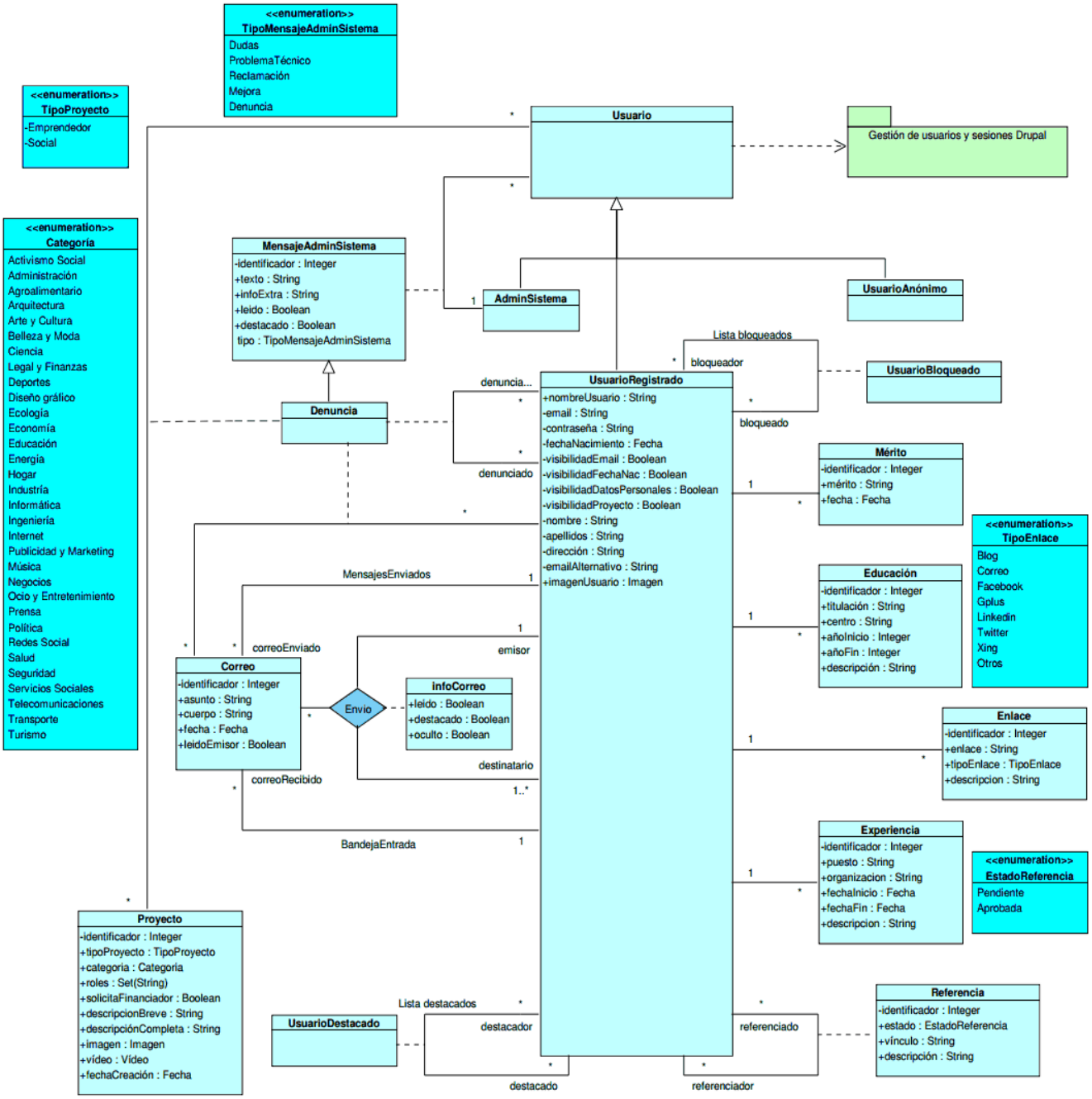


Diagrama de clase de Grupo, Recaudación y Objetivos

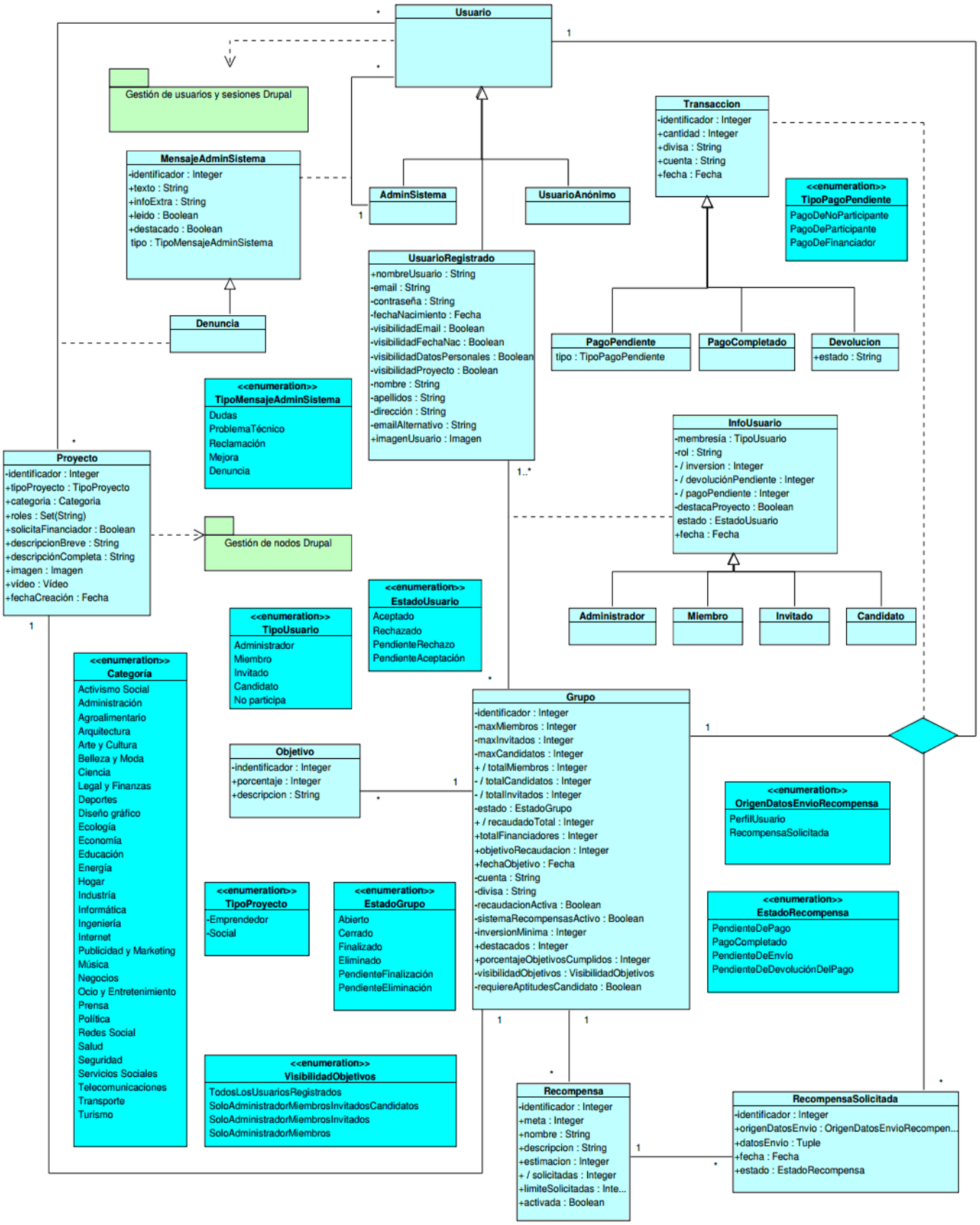


Diagrama de clase de Asociación con Grupo

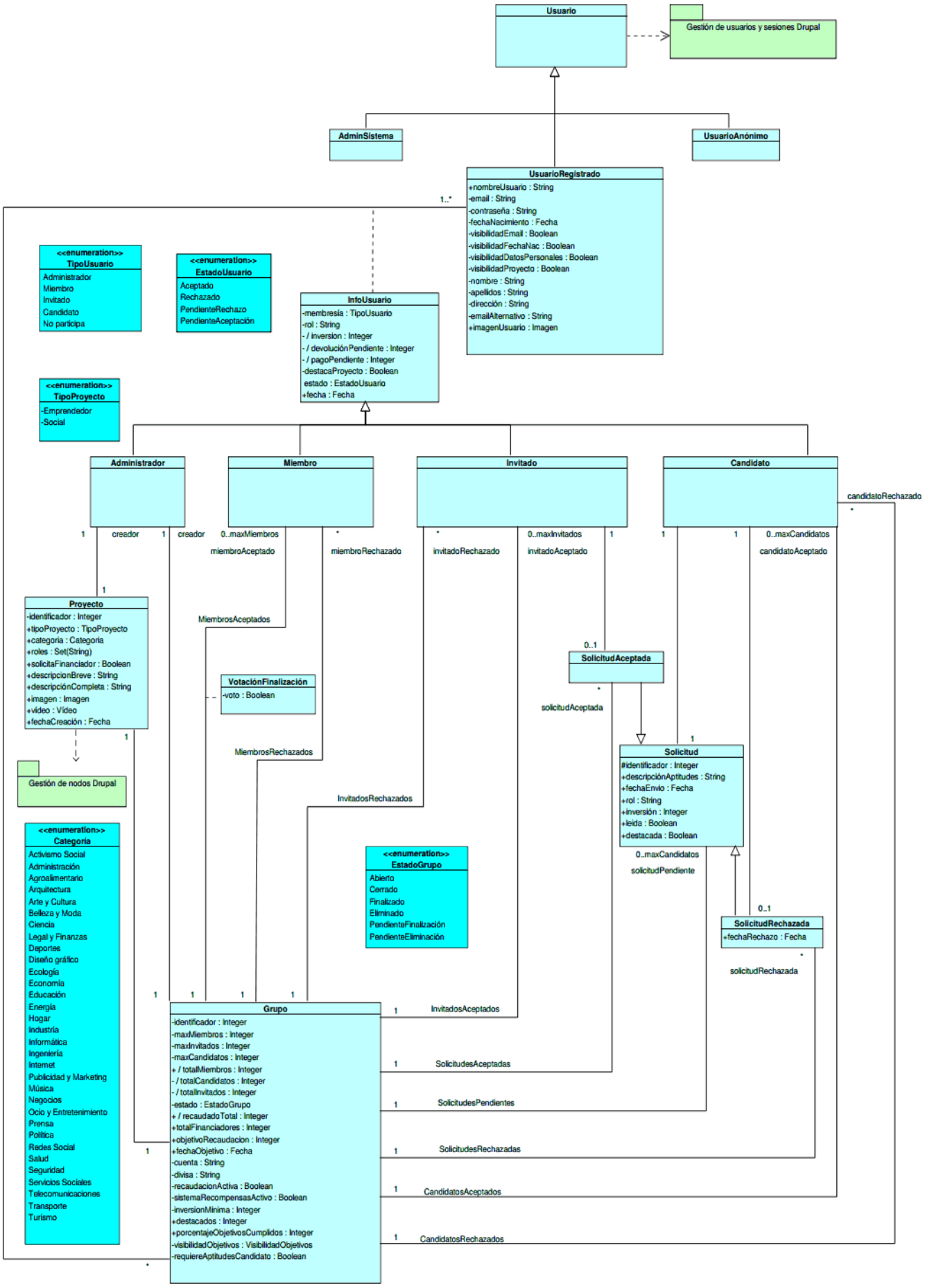


Diagrama de clase de Chat

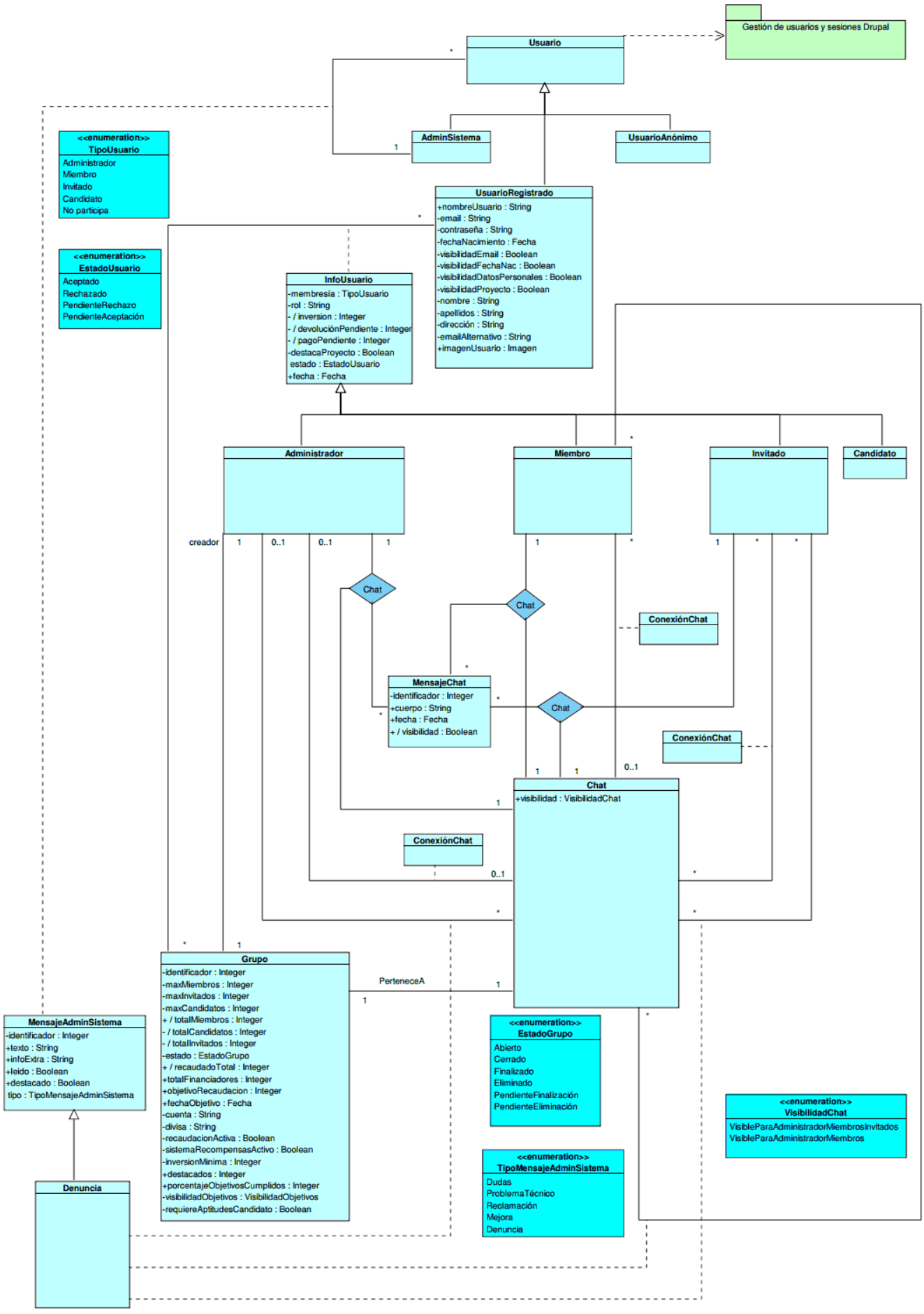


Diagrama de clase de Muro

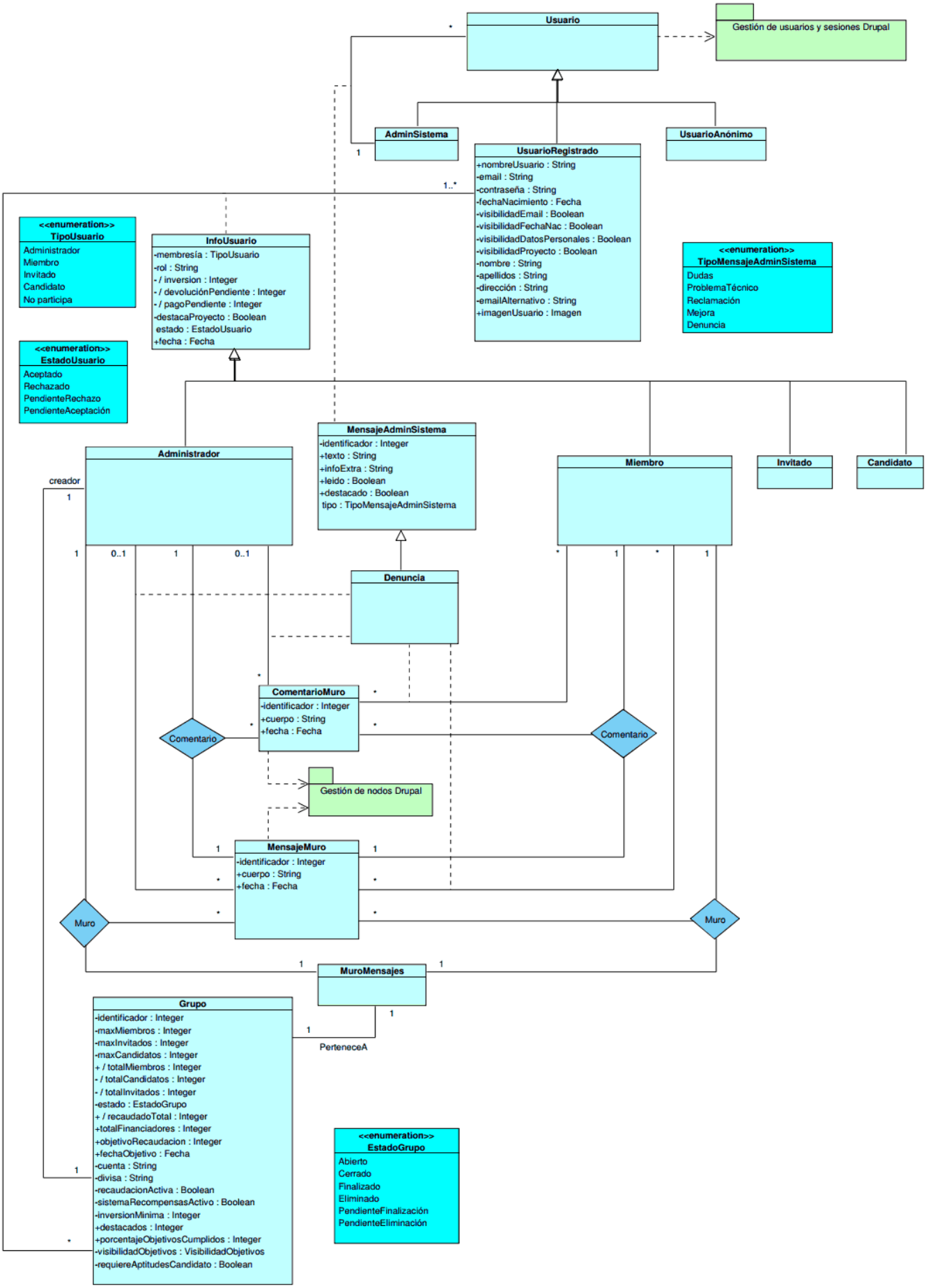
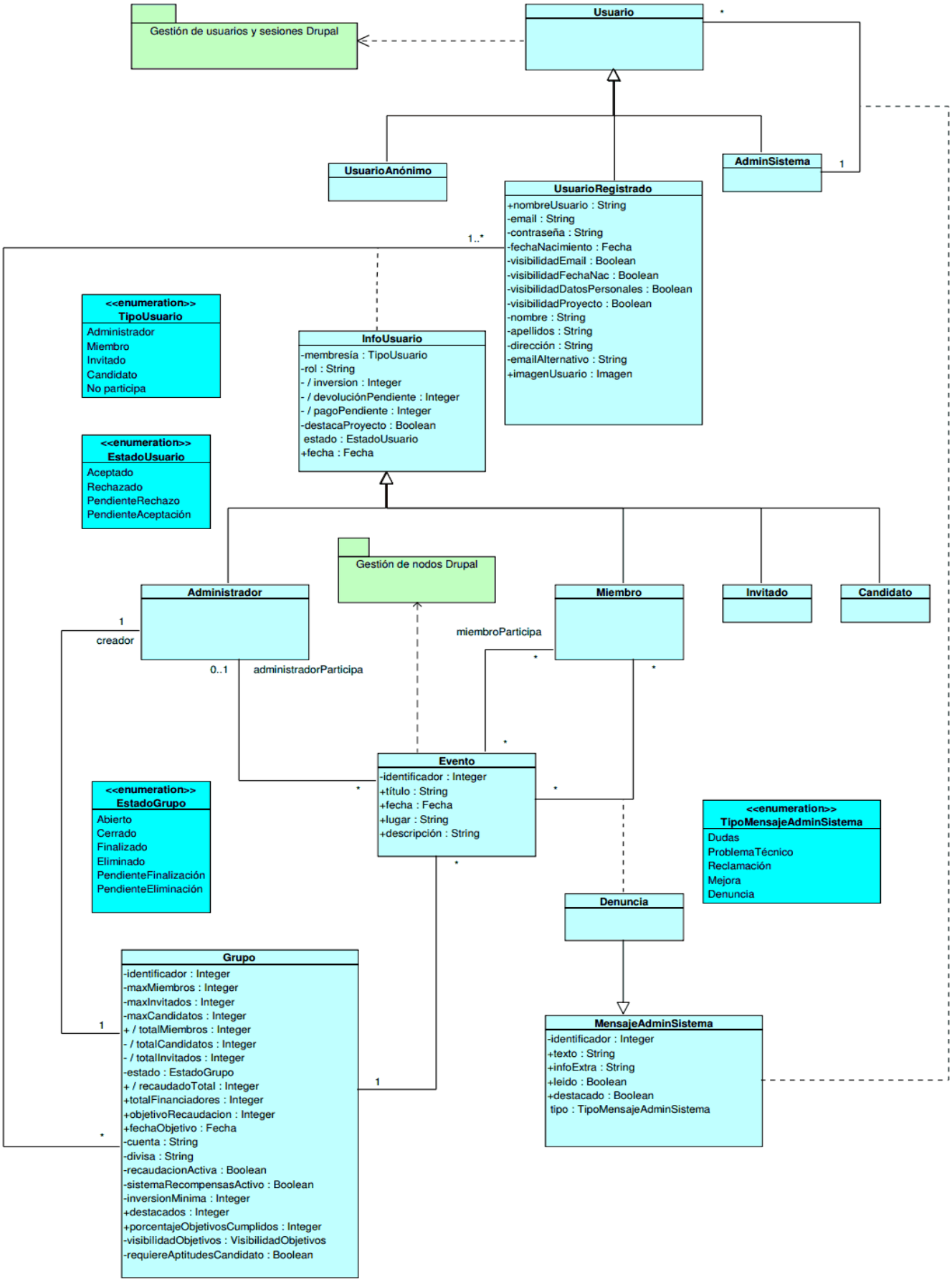


Diagrama de clase de Eventos

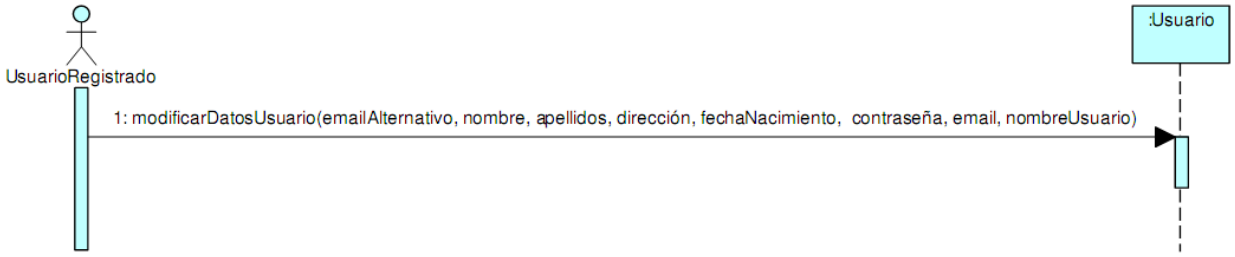


Apéndice II – Diagramas de secuencia

1. Diagramas de secuencia de usuario

Modificar datos de usuario

Introduce o modifica los datos que el resto de usuarios verán en el perfil, salvo la contraseña y el email con el que inicia sesión. Tanto el emailAlternativo, como nombre, apellios, y dirección pueden se utilizados a la hora de solicitar una recompensa como datos de envío para esta.



Operación: modificarDatosUsuario(emailAlternativo : String, nombre : String, apellidos : String, dirección : String, fechaNacimiento: Fecha, contraseña : String, email : String, nombreUsuario : String)
Descripción: Modifica los datos que el usuario solicite modificar
Precondiciones:

En caso de indicar una contraseña ha de tener una longitud de entre 8 y 30 caracteres
```
if not contraseña -> oclIsUndefined() then
    contraseña -> size() >= 8 and contraseña -> size() <= 30
endif
```

En caso de indicar un email alternativo ha de tener una longitud de entre 1 y 64 caracteres
```
if not emailAlternativo -> oclIsUndefined() then
    emailAlternativo -> size() >= 1 and emailAlternativo -> size() <= 64
endif
```

En caso de indicar el nombre ha de tener una longitud de entre 1 y 25 caracteres
```
if not nombre -> oclIsUndefined() then
    nombre -> size() >= 1 and nombre -> size() <= 25
endif
```

En caso de indicar los apellidos han de tener una longitud de entre 1 y 50 caracteres
```
if not apellidos -> oclIsUndefined() then
    apellidos -> size() >= 1 and apellidos -> size() <= 50
endif
```

En caso de indicar una dirección ha de tener una longitud de entre 1 y 100 caracteres
```
if not dirección -> oclIsUndefined() then
    dirección -> size() >= 1 and dirección -> size() <= 100
endif
```

En caso de indicar una fecha de nacimiento tiene que ser al menos de 14 años
```
if not fechaNacimiento -> oclIsUndefined() then
    fechaActual() - fechaNacimiento -> formatoFecha('Año') >= 14
endif
```

En caso de indicar el email ha de tener una longitud de entre 1 y 64 caracteres
```
if not email -> oclIsUndefined() then
    email -> size() >= 1 and email -> size() <= 64
endif
```

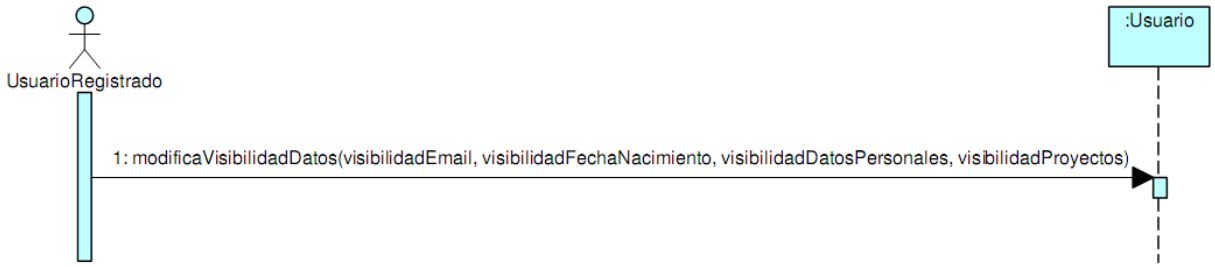
En caso de indicar el nombre de usuario ha de tener una longitud de entre 1 y 25 caracteres
```
if not nombreUsuario -> oclIsUndefined() then
    nombreUsuario -> size() >= 1 and nombreUsuario -> size() <= 25
endif
```

Postcondiciones:
Modifica los datos que el usuario haya solicitado modificar

```
if not emailAlternativo -> oclIsUndefined() then
  self.emailAlternativo = emailAlternativo
endif
if not nombre -> oclIsUndefined() then
  self.nombreReal = nombre
endif
if not apellidos -> oclIsUndefined() then
  self.emailAlternativo = emailAlternativo
endif
if not dirección -> oclIsUndefined() then
  self.dirección = dirección
endif
if not fechaNacimiento -> oclIsUndefined() then
  self.fechaNacimiento = fechaNacimiento
endif
if not contraseña-> oclIsUndefined() then
  self.contraseña = contraseña
endif
if not email > oclIsUndefined() then
  self.email = email
endif
if not nombreUsuario -> oclIsUndefined() then
  self.nombreUsuario = nombreUsuario
endif
```

Modificar visibilidad datos usuario

El usuario puede modificar los datos que quiere que sean visibles para el resto de usuarios registrados. Solo siendo posible visibilizar u ocultar, el email alternativo, que puede coincidir o no con el utilizado para iniciar sesión, edad y/o fecha de nacimiento y proyectos donde participa como administrador o miembro. Además es posible modificar la visibilidad de los datos personales, los cuales son utilizados también en el momento de pedir recompensas.

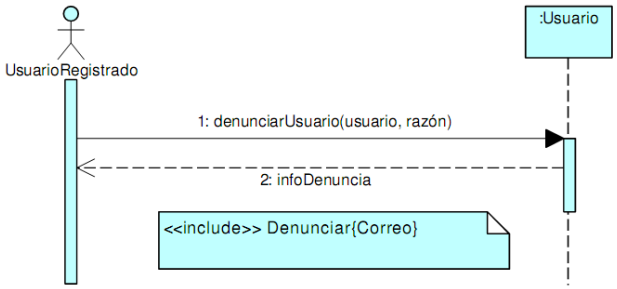


Operación: modificaVisibilidadDatos(visibilidadEmail: Boolean, visibilidadFechaNacimiento: Boolean, visibilidadDatosPersonales: Boolean, visibilidadProyectos: Boolean)
Descripción: Modifica la visibilidad de sus propios datos
Precondiciones: -
Postcondiciones:
La visibilidad de sus datos queda guardada en el sistema

```
if not visibilidadEmail -> oclIsUndefined() then
  self.visibilidadEmail = visibilidadEmail
endif
if not visibilidadFechaNacimiento -> oclIsUndefined() then
  self.visibilidadFechaNac = visibilidadFechaNacimiento
endif
if not visibilidadDatosPersonales -> oclIsUndefined() then
  self.visibilidadDatosPersonales = visibilidadDatosPersonales
endif
if not visibilidadProyectos -> oclIsUndefined() then
  self.visibilidadProyectos = visibilidadProyectos
endif
```

Denunciar usuario

El usuario denuncia a otro usuario registrado que no sea el mismo indicando una razón.



Operación: denunciarUsuario(usuario: UsuarioRegistrado, razón: String) : TupleType(razón: String, infoExtra: String)
Descripción: Obtiene los datos necesarios para formalizar la denuncia.
Precondiciones:
Debe existir el usuario en el sistema
Usuario.allInstances() -> exists(u | u = usuario)

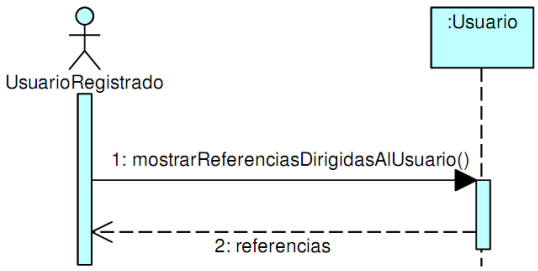
El usuario a denunciar no es él mismo
usuario <> self

Salida:
Se recopila la información necesaria para enviar.

```
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{razón = razón,
        infoExtra = 'Tipo contenido: Usuario, IdentificadorUsuario: '  -> concat(usuario.identificador)}
```

Ver referencias dirigidas al usuario

Muestra las referencias que otros usuarios registrados han escrito al usuario y que este ha aprobado. Estas referencias se componen de información sobre en que están relacionados ambos usuarios, tal como profesor de, alumno de, empleado de,... y una descripción de la referencia.

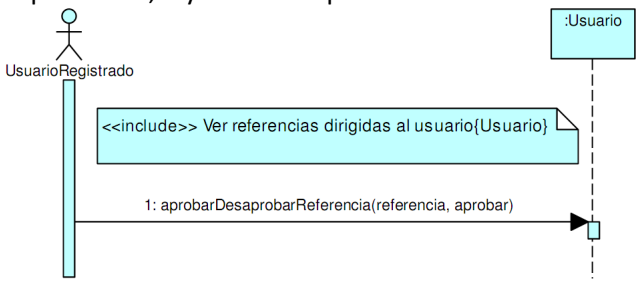


Operación: mostrarReferenciasDirigidasAlUsuario() : Set(Referencia)
Descripción: Muestra las referencias creadas para el usuario.
Precondiciones: -
Salida:
Genera el listado de referencias que otros usuarios han creado para él.

```
let referencias : Set(Referencia) = self.referencia -> select(r | r.referenciado = self)
```

Ver referencias dirigidas al usuario

Muestra las referencias que otros usuarios registrados han escrito al usuario y que este ha aprobado. Estas referencias se componen de información sobre en que están relacionados ambos usuarios, tal como profesor de, alumno de, empleado de,... y una descripción de la referencia.



Operación: aprobarDesaprobarReferencia(referencia: Referencia, aprobar: Boolean)
Descripción: Muestra las referencias creadas para el usuario
Precondiciones:

La referencia existe en el sistema
Referencia.allInstances() -> exists(r | r = referencia)

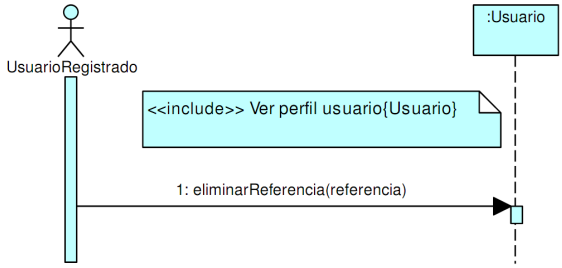
La referencia va dirigida al Usuario Registrado
referencia.referenciado = self

Postcondiciones:
La referencia selecciona es aprobada o desaprobada según lo indicado por el usuario

if aprobar then
referencia.estado = EstadoReferencia::Aprobada
else
referencia.referenciador -> oclIsUndefined() and referencia.referenciado -> oclIsUndefined() and
Referencia.allInstances() -> excludes(referencia)

Eliminar referencia

Elimina una referencia dirigida al usuario propietario del perfil.



Operación: eliminarReferencia(referencia: Referencia)
Descripción: Elimina una referencia dirigida a otro usuario
Precondiciones:

La referencia existe en el sistema
Referencia.allInstances() -> exists(r | r = referencia)

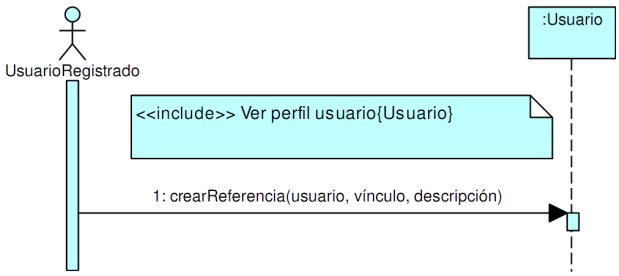
La referencia es del usuarioRegistrado
referencia.referenciador = self

Postcondiciones:
La referencia queda eliminada del sistema

referencia.referenciador -> oclIsUndefined() and referencia.referenciado -> oclIsUndefined() and
Referencia.allInstances() -> excludes(referencia)

Escribir referencia

Escribe una referencia dirigida al usuario propietario del perfil siempre y cuando no tenga ninguna ya escrita por el usuario visitante del perfil. Estas referencias está compuesta de información sobre en que están relacionados ambos usuarios, tal como profesor de, alumno de, empleado de,... y una descripción de la referencia.



Operación: crearReferencia(usuario: UsuarioRegistrado, vínculo: String, descripción: String)
Descripción: Escribe una referencia dirigida hacia otro usuario
Precondiciones:

El usuario existe en el sistema
UsuarioRegistrado.allInstances() -> exists(u | u = usuario)

El usuario referenciado no tiene ya una referencia del usuario referenciador
Referencia.allInstances() -> select(r | r.referenciado = usuario and r.referenciador = self) -> isEmpty()

El vínculo ha de tener una longitud de entre 1 y 50 caracteres
vínculo -> size() >= 1 and vínculo -> size() <= 50

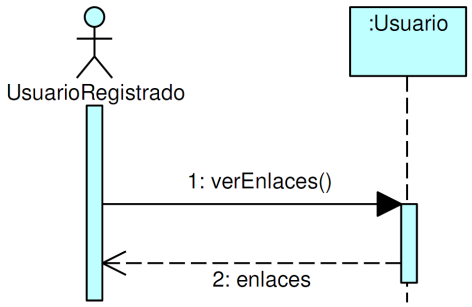
La descripción ha de tener una longitud de entre 1 y 300 caracteres
descripción -> size() >= 1 and descripción -> size() <= 300

Postcondiciones:
La nueva referencia es creada, teniendo como referenciador al usuario que la escribe y como referenciado al usuario al que va dirigida. La referencia estará en estado pendiente de ser aprobada o eliminada por el usuario referenciado.

let referencia: Referencia -> oclIsUndefined() in
referencia.oclIsNew() and referencia.oclIsTypeOf(Referencia) and referencia.referenciador = self and
referencia.referenciado = usuario and referencia.vínculo = vínculo and referencia.descripción = descripción and
referencia.estado = EstadoReferencia::Pendiente

Ver enlaces

Muestra los enlaces que el propio usuario a añadido. Estos enlaces son útiles para su perfil público, para que puedan tener más fuentes de información donde conocer al usuario y así este poder ganar reputación.

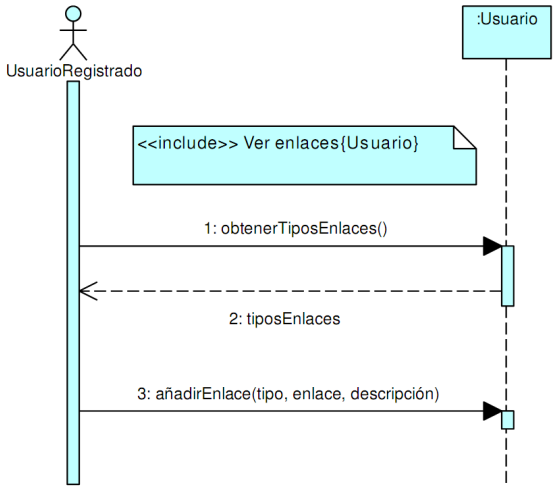


Operación: verEnlaces() : Set(Enlace)
Descripción: Muestra los enlaces web del usuario
Precondiciones: -
Salida:
Genera el listado de enlaces web del usuario

let enlaces : Set(Enlace) = self.enlace

Añadir enlace

Añade un nuevo enlace al listado de enlaces. Este enlace está compuesto por un tipo ya sea Blog, Twitter, Facebook, etc., el enlace web en sí y una descripción.



Operación: obtenerTiposEnlaces() : Set(TipoEnlace)
Descripción: Añade un nuevo enlace web al perfil del usuario
Precondiciones: -
Salida:
Genera el listado de tipos de enlace disponibles

```
let tiposEnlaces: Set(TipoEnlace) = TipoEnlace.allInstances()
```

Operación: añadirEnlace(tipo: TipoEnlace, direcciónWeb: String, descripción: String)
Descripción: Añade un nuevo enlace web al perfil del usuario
Precondiciones:
No existe otro enlace del usuario con mismo enlace web

```
self.enlace -> select(e | e.enlace = direcciónWeb)
```

La dirección web debe ser un string entre 1 y 2000 caracteres y la descripción, en caso de ser indicada, un string de entre 1 y 100 caracteres.

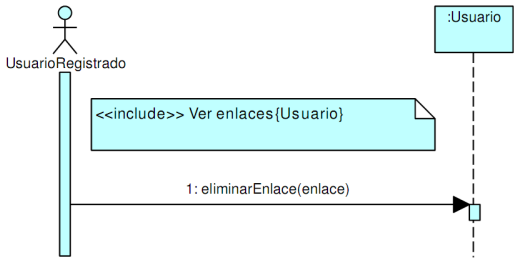
```
not direcciónWeb -> oclIsUndefined() and direcciónWeb -> size() >= 1 and direcciónWeb -> size() <= 2000 and
(descripción -> oclIsUndefined() or (not descripción -> oclIsUndefined and descripción >= 1 and descripción <= 100))
```

Postcondiciones:
Añade un nuevo enlace web al listado de enlaces del usuario

```
let enlace: Enlace -> oclIsUndefined() in
enlace.oclIsNew() and enlace.oclIsTypeOf(Enlace) and enlace.tipo = tipo and enlace.enlace = direcciónWeb and
self.enlace -> includes(enlace)

if not descripción -> oclIsUndefined() then
    enlace.descripción = descripción
endif
```

Eliminar enlace

Eliminar un enlace del perfil del propio usuario.



Operación: eliminarEnlace(enlace: Enlace)
Descripción: Elimina un enlace del usuario
Precondiciones:

El enlace existe en el sistema
Enlace.allInstances() -> exists(e | e = enlace)

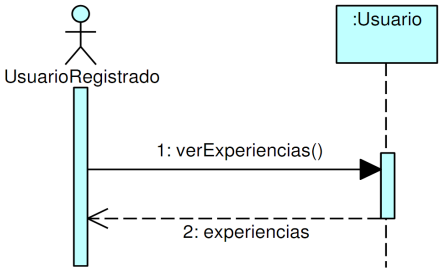
El enlace es del usuarioRegistrado
self.enlace -> includes(enlace)

Postcondiciones:
El enlace queda eliminado del sistema

self.enlace -> excludes(enlace) and Enlace.allInstances() -> excludes(enlace)

Ver experiencia

Muestra las experiencias que el propio usuario a añadido. Estas experiencias son útiles para su perfil público, demostrando como ha desarrollado sus habilidades participando en empresas u organizaciones.

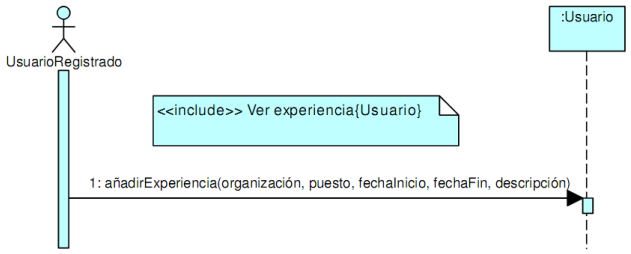


Operación: verExperiencias() : Set(Experiencia)
Descripción: Muestra la experiencia en empresas y/o organizaciones del usuario
Precondiciones: -
Salida:
Genera el listado de sus experiencias profesionales

let experiencias: Set(Experiencia) = self.experiencia

Añadir experiencia

Añade una nueva experiencia al listado de experiencias del usuario. Esta experiencia está compuesta por la empresa u organización en la que trabajó o participó, el puesto o actividad que desempeñó, la fecha de inicio y fin del periodo en el que estuvo en la organización y una descripción de los proyectos que realizó.



Operación: añadirExperiencia(organización:String, puesto:String, fechaInicio: Fecha, fechaFin:Fecha, descripción:String)
Descripción: Añade una nueva experiencia profesional en el perfil del usuario
Precondiciones:

La organización, puesto, fechaInicio y fechaFin no pueden estar vacíos
not organización -> oclIsUndefined() and not puesto -> oclIsUndefined() and not fechaInicio -> oclIsUndefined() and not fechaFin -> oclIsUndefined()

La organización y el puesto deben ser strings entre 1 y 60 caracteres
organización -> size() >= 1 and organización -> size() <= 60 and puesto -> size() >= 1 and puesto -> size() <= 60

La descripción, en caso de ser indicada, debe ser un string de entre 1 y 300 caracteres.
descripción -> oclIsUndefined() or (not descripción -> oclIsUndefined and
descripción -> size() >= 1 and descripción -> size() <= 300)

La fecha de inicio ha de ser menor a la fecha fin
fechaInicio < fechaFin

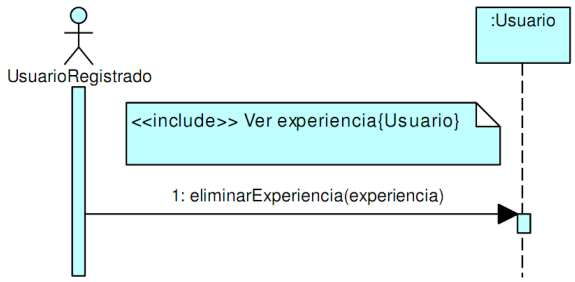
Postcondiciones:
Añade una nueva experiencia profesional al listado de experiencias del usuario

let experiencia: Experiencia -> oclIsUndefined() in
experiencia.oclIsNew() and experiencia.oclIsTypeOf(Experiencia) and experiencia.organización = organización and
experiencia.puesto = puesto and experiencia.fechaInicio = fechaInicio and experiencia.fechaFin = fechaFin and
self.experiencia -> includes(experiencia)

if not descripción -> oclIsUndefined() then
experiencia.descripción = descripción
endif

Eliminar experiencia

Eliminar una experiencia del perfil del propio usuario.



Operación: eliminarExperiencia(experiencia: Experiencia)
Descripción: Elimina una experiencia profesional del usuario
Precondiciones:

La experiencia existe en el sistema
Experiencia.allInstances() -> exists(e | e = experiencia)

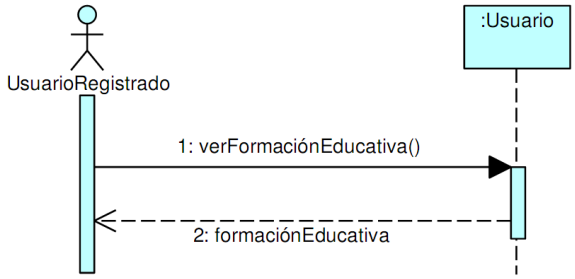
La experiencia es del usuarioRegistrado
self.experiencia -> includes(experiencia)

Postcondiciones:
La experiencia queda eliminada del sistema

self.experiencia -> excludes(experiencia) and
Experiencia.allInstances() -> excludes(experiencia)

Ver formación educativa

Muestra las formaciones educativas que el propio usuario a añadido. Estas formaciones son útiles para su perfil público, demostrando qué tipo de conocimientos ha adquirido y como de capacitado está para colaborar en los proyectos.

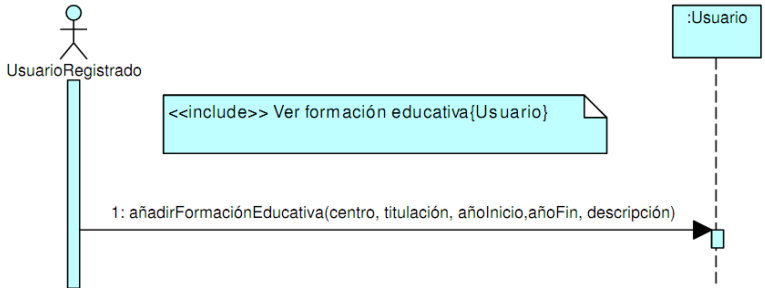


Operación: verFormaciónEducativa() : Set(Educación)
Descripción: Muestra la formación educativa del usuario
Precondiciones: -
Salida:
Genera el listado de formaciones educativas del usuario

let formaciónEducativa : Set(Educación) = self.educación

Añadir formación educativa

Añade una nueva formación al listado de formaciones educativas del usuario. Esta formación está compuesta por el centro en el que se formó, la titulación que consiguió, el año de inicio y fin del periodo en el que estuvo en el centro y una descripción de la titulación o los conocimientos que adquirió.



Operación: añadirFormaciónEducativa(centro:String, titulación:String, añoInicio:Integer, añoFin:Integer, descripción:String)
Descripción: Añade una nueva formación educativa en el perfil del usuario
Precondiciones:

El centro, titulación, añoInicio y añoFin no pueden estar vacíos
not centro -> oclIsUndefined() and not titulación -> oclIsUndefined() and not añoInicio -> oclIsUndefined() and not añoFin -> oclIsUndefined()

El centro y titulación deben ser strings entre 1 y 60 caracteres
centro -> size() >= 1 and centro -> size() <= 60 and titulación -> size() >= 1 and titulación -> size() <= 60

La descripción, en caso de ser indicada, debe ser un string de entre 1 y 300 caracteres.
descripción -> oclIsUndefined() or (not descripción -> oclIsUndefined and
descripción -> size() >= 1 and descripción -> size() <= 300)
El año de inicio ha de ser menor a que el año fin
añoInicio < añoFin

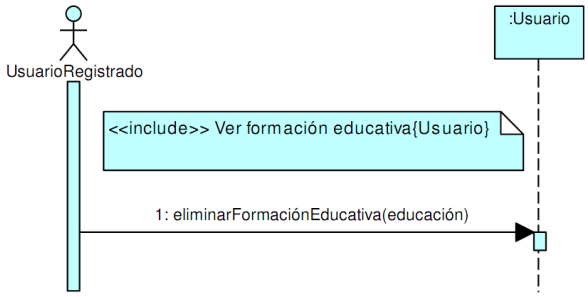
Postcondiciones:
Añade una nueva formación educativa al listado de formaciones del usuario

let educación: Educación -> oclIsUndefined() in
educación.oclIsNew() and educación.oclITypeOf(Educación) and educación.centro = centro and
educación.titulación = titulación and educación.añoInicio = añoInicio and educación.añoFin = añoFin and
self.educación -> includes(educación)

if not descripción -> oclIsUndefined() then
educación.descripción = descripción
endif

Eliminar formación educativa

Elimina una formación educativa del perfil del propio usuario.



Operación: eliminarFormaciónEducativa(educación: Educación)
Descripción: Elimina una formación educativa del usuario
Precondiciones:

La formación educativa existe en el sistema
Educación.allInstances() -> exists(e | e = educación)

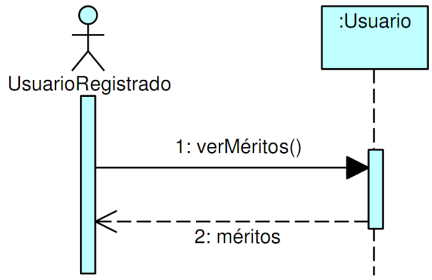
La formación es del usuarioRegistrado
self.educación -> includes(educación)

Postcondiciones:
La formación queda eliminada del sistema

self.educación -> excludes(educación) and
Educación.allInstances() -> excludes(educación)

Ver méritos

Muestra los méritos o premios conseguidos por el propio usuario. Estas méritos son útiles para su perfil público, demostrando el reconocimiento que otros profesionales de su campo le otorgan.

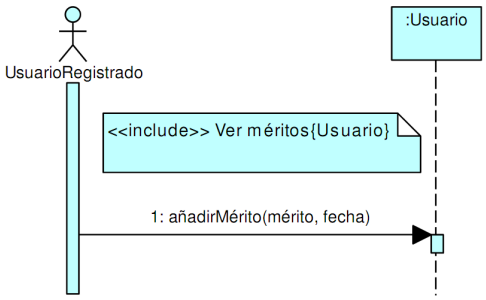


Operación: verMéritos() : Set(Mérito)
Descripción: Muestra los méritos o premios conseguidos por el usuario
Precondiciones: -
Salida:
Genera el listado de méritos o premios conseguidos por el usuario

let méritos: Set(Mérito) = self.mérito

Añadir mérito

Añade un nuevo mérito al listado de méritos del usuario. Este mérito está compuesta por el nombre del mismo y la fecha en que fue otorgado.



Operación: añadirMérito(títuloMérito: String, fecha: Fecha)
Descripción: Añade un nuevo mérito en el perfil del usuario
Precondiciones:

El mérito y fecha no pueden estar vacíos
not títuloMérito -> oclIsUndefined() and not fecha -> oclIsUndefined()

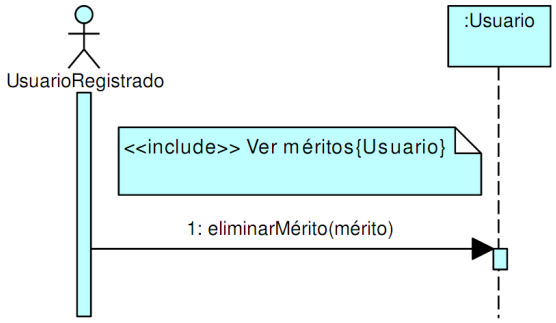
El mérito debe ser un string de entre 1 y 50 caracteres
títuloMérito -> size() >= 1 and títuloMérito -> size() <= 50

Postcondiciones:
Añade un nuevo mérito al listado de méritos del usuario

let mérito: Mérito -> oclIsUndefined() in
mérito.oclIsNew() and mérito.oclIsTypeOf(Mérito) and
mérito.mérito = títuloMérito and mérito.fecha = fecha and
self.mérito -> includes(mérito)

Eliminar mérito

Elimina un mérito del perfil del propio usuario.



Operación: eliminarMérito(mérito: Mérito)
Descripción: Elimina un mérito del usuario
Precondiciones:

El mérito existe en el sistema
Mérito.allInstances() -> exists(m | m = mérito)

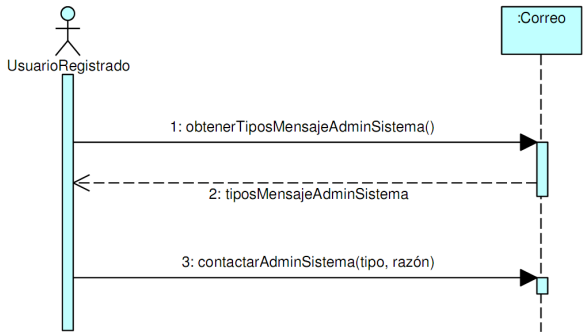
El mérito es del usuarioRegistrado
self.mérito -> includes(mérito)
Postcondiciones:
El mérito queda eliminado del sistema

self.mérito -> excludes(mérito) and
Mérito.allInstances() -> excludes(mérito)

2. Diagramas de secuencia de correo

Contactar AdminSistema

Contacta con el administrador del sistema enviándole un mensaje de un tipo concreto además de añadirle una descripción del asunto. Este tipo puede ser de mejoras, dudas, problema técnico, reclamación o denuncia. Una vez revisado lo que el usuario ha enviado, el administrador del sistema procedería a realizar las acciones necesarias y contestarle con un correo en su bandeja de entrada.



Operación: obtenerTiposMensajeAdminSistema(): Set(TipoMensajeAdminSistema)
Descripción: Obtiene los tipos de mensaje a enviar al administrador del sistema posibles.
Precondiciones: -

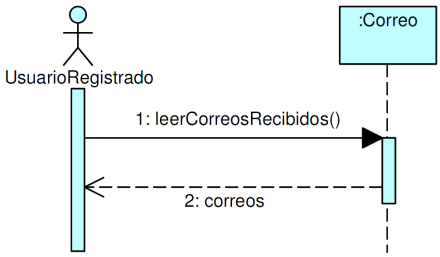
Salida:
let tiposMensajeAdminSistema : Set(TipoMensajeAdminSistema) = TipoMensajeAdminSistema.allInstances()

Operación: contactarAdminSistema(tipo: TipoMensajeAdminSistema, razón)
Descripción: Envía un mensaje al Administrador del sistema indicando un tipo, siendo los posibles, dudas, problema Técnico, reclamaciones, mejoras, denuncia
Precondiciones: -
PostCondiciones:
El mensaje es enviado al Administrador del sistema para posterior revisión.

let mensajeAdminSistema : MensajeAdminSistema -> oclIsUndefined() in
mensajeAdminSistema.oclIsNew() and mensajeAdminSistema.oclIsTypeOf(MensajeAdminSistema) and
mensajeAdminSistema.fecha = fechaActual() and mensajeAdminSistema.texto = razón and
mensajeAdminSistema.infoExtra = '' and mensajeAdminSistema.tipo = tipo and
mensajeAdminSistema.identificador = MensajeAdminSistema.allInstances().identificador -> max() + 1 and
mensajeAdminSistema.leido = false and mensajeAdminSistema.destacado = false

Leer correo recibidos

Muestra el listado de correos recibidos, en los que el usuario es uno de los destinatarios.

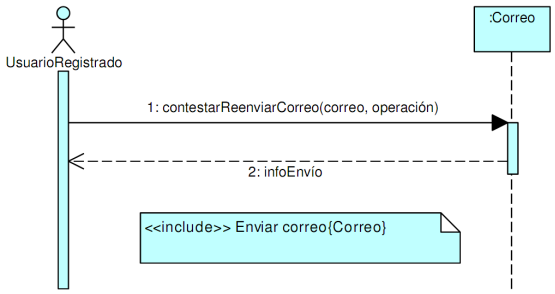


Operación: leerCorreosRecibidos(): Set(Correo)
Descripción: El usuario lee los correos que le han sido enviados
Precondiciones: -
Salida:
Muestra un listado de correos donde el usuario es el destinatario

```
let correos : Set(Correo) = self.correoRecibido
```

Contestar/Reenviar correo

Contesta o reenvia un correo. Si es contestado creará un correo listo para enviar donde el emisor aparezca como destinatario, el asunto será el mismo pero precedido de 'Re: ' y el cuerpo será el mismo del original pero añadiendo la fecha y emisor originales. En caso de ser reenviado, el destinatario y asunto aparecerán vacíos, pero en el cuerpo aparecerán tanto la fecha, emisor, destinatario, asunto y cuerpo del correo original.



Operación: contestarReenviarCorreo(correo: Correo, operación: String) : TupleType(destinatarios: Set(UsuarioRegistrado), asunto: String, cuerpo: String)

Descripción: El correo es enviado a los destinatarios
Precondiciones:

El correo debe existir en el sistema

```
Correo.allInstances() -> exists(c | c = correo)
```

La operación es igual a 'contestar' o 'reenviar'

```
operación = 'contestar' or operación = 'reenviar'
```

El usuario es uno de los destinatarios o el emisor del correo

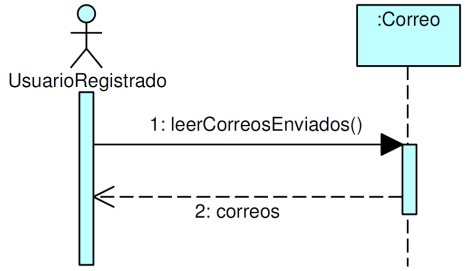
```
correo.infoCorreo.emisor = self or correo.infoCorreo.destinatario -> includes(self)
```

Postcondiciones:
Los campos de destinatarios, asunto y cuerpo son preparados para el envío.

```
let infoEnvio : TupleType(destinatarios: Set(UsuarioRegistrado),asunto: String, cuerpo: String) -> oclIsUndefined() in
if operación = 'contestar' then
  infoEnvio.destintarios -> includes(correo.infoCorreo.emisor) and infoEnvio.asunto = 'Re: ' ->concat(correo.asunto)
  and infoEnvio.cuerpo = correo.fecha -> concat(', de ') -> concat(correo.infoCorreo.emisor.nombreUsuario) ->
  concat(' escribió: ') -> concat(correo.cuerpo)
elseif operación = 'reenviar' then
  infoEnvio.destintarios -> oclIsUndefined() and infoEnvio.asunto -> oclIsUndefined() and
  infoEnvio.cuerpo = '---Mensaje reenviado---'
                    -> concat('De: ') -> concat(correo.infoCorreo.emisor.nombreUsuario)
                    -> concat('Fecha: ') -> concat(correo.fecha)
                    -> concat('Asunto: ') -> concat(correo.asunto)
                    -> concat('Para: ') and
  correo.infoCorreo.destinatario -> iterate(d | infoEnvio.cuerpo = infoEnvio.cuerpo -> concat(d.nombreUsuario)
endif
```

Leer correo enviados

Muestra el listado de correos enviados, en los que el usuario es el emisor.

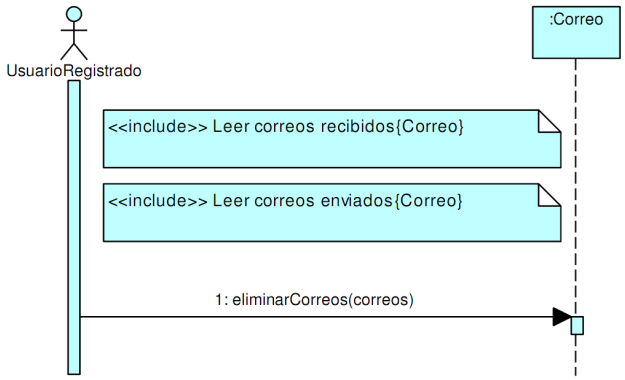


Operación: leerCorreosEnviados(): Set(Correo)
Descripción: El usuario lee los correos que ha enviado
Precondiciones: -
Salida:
Muestra un listado de correos donde el usuario es el emisor

```
let correos : Set(Correo) = self.correoEnviado
```

Eliminar correos

Elimina un conjunto de correos del usuario, siendo todos o recibidos o enviados por él.



Operación: eliminarCorreos(correos: Set(Correo))
Descripción: Elimina correos seleccionados
Precondiciones:

Todos los correos existen

```
Correo.allInstances() -> includesAll(correos)
```

Todos los correos o son recibidos o enviados y no ambos

```
self.correoRecibido -> includesAll(correos) or self.correoEnviado -> includesAll(correos)
```

Si los correos son recibidos el usuario estará entre los destinatarios de cada uno de ellos,
si es de los enviados será el emisor de todos.

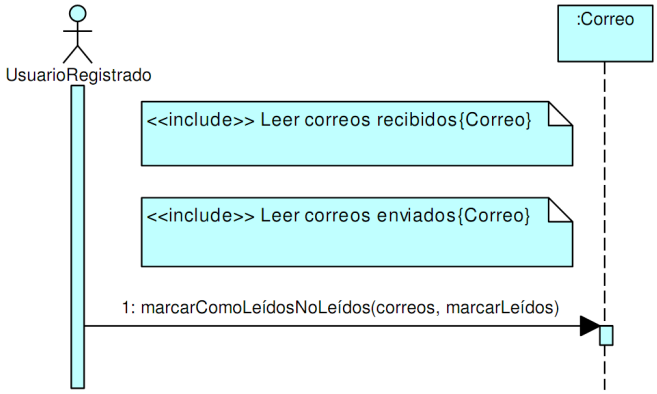
```
if self.correoRecibido -> includesAll(correos) then
    correos -> forAll(c | c.infoCorreo.destinatario -> exists(d | d = self)
elseif self.correoEnviado -> includesAll(correos) then
    correos -> forAll(c | c.infoCorreo.emisor = self)
endif
```

Postcondiciones:
Los correos seleccionados son eliminados del sistema

```
if self.correoRecibido -> includesAll(correos) then
    self.correoRecibido -> excludesAll(correos)
    InfoUsuario.allInstances() -> excludes(correos.infoUsuario -> select(i | i.destinatario = self))
elseif self.correoEnviado -> includesAll(correos) then
    self.correoEnviado -> excludesAll(correos)
    InfoUsuario.allInstances() -> excludes(correos.infoUsuario -> select(i | i.emisor = self))
endif
Correo.allInstances() -> excludes(correos -> select(c | c.infoUsuario -> oclIsUndefined())
```

Marcar correos como leídos o no leídos

Marca un conjunto de correos o enviados o recibidos como leídos o no leídos.



Operación: marcarComoLeídosNoLeídos(correos: Set(Correo), marcarLeídos : Boolean)
Descripción: Marca correos como leídos o no leídos
Precondiciones:

Todos los correos existen
```
Correo.allInstances() -> includesAll(correos)
```

Todos los correos o son recibidos o enviados y no ambos
```
self.correoRecibido -> includesAll(correos) or self.correoEnviado -> includesAll(correos)
```

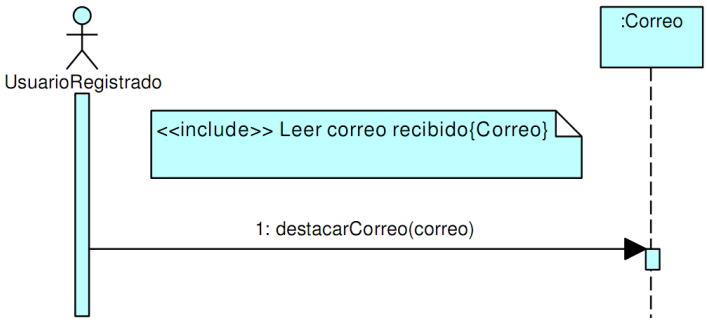
Si los correos son recibidos el usuario estará entre los destinatarios de cada uno de ellos, si es de los enviados será el emisor de todos
```
if self.correoRecibido -> includesAll(correos) then
    correos -> forAll(c | c.infoCorreo.destinatario -> exists(d | d = self))
elseif self.correoEnviado -> includesAll(correos) then
    correos -> forAll(c | c.infoCorreo.emisor = self)
endif
```

Postcondiciones:
Los correos seleccionados son marcados como leídos o no leídos

```
if self.correoRecibido -> includesAll(correos) then
    correos.infoCorreo -> select(i | i.destinatario = self) -> forAll(i | i.leido = marcarLeídos)
elseif self.correoEnviado -> includesAll(correos) then
    correos -> forAll(c | c.leidoEmisor = marcarLeídos)
endif
```

Destacar correo

Destaca o deja de destacar un correo recibido del usuario.



Operación: destacarCorreo(correo: Correo)
Descripción: Destacar un correo
Precondiciones:

El correo existe en el sistema
Correo.allInstances() -> includes(correo)

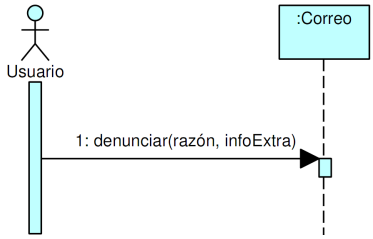
El correo es recibido y el usuario es uno de los destinatarios
self.correoRecibido -> includes(correo) and correo.infoCorreo.destinatario -> includes(self)

Postcondiciones:
El correo seleccionado queda destacado si no lo estaba o deja de estar destacado si ya lo estaba.

```
let infoCorreo : InfoCorreo = correo.infoCorreo -> select(i | i.destinatario = self) in
if infoCorreo.destacado then
    infoCorreo.destacado = false
else
    infoCorreo.destacado = true
endif
```

Denunciar

Crea un mensaje de tipo denuncia y lo envía al administrador del sistema junto con la descripción de la denuncia. Una vez revisado lo que el usuario ha enviado, el administrador del sistema procedería a realizar las acciones necesarias y contestarle con un correo en su bandeja de entrada.

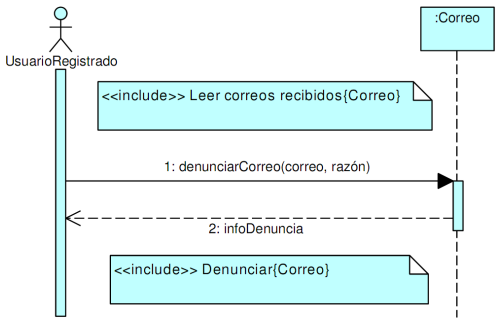


Operación: denunciar(asunto: String, razón: String, infoExtra: String)
Descripción: Denuncia contenido ofensivo
Precondiciones: -
PostCondiciones:
La denuncia es creada y enviada al Administrador del sistema.

```
denuncia.oclIsNew() and denuncia.oclIsTypeOf(Denuncia) and
denuncia.fecha = fechaActual() and denuncia.texto = razón and
denuncia.infoExtra = infoExtra and
denuncia.identificador = Denuncia.allInstances().identificador -> max() + 1 and
denuncia.leido = false and denuncia.destacado = false and
denuncia.tipo = TipoMensajeAdminSistema::Denuncia and
denuncia.usuario = self
```

Denunciar correo

El usuario denuncia un correo recibido indicando una razón.



Operación: denunciarCorreo(correo: Correo, razón: String) :
TupleType(razón: String, infoExtra: String)
Descripción: Obtiene los datos necesarios para formalizar la denuncia
Precondiciones:
Debe existir el correo en el sistema
Correo.allInstances() -> exists(c | c = correo)

El correo debe haber sido recibido y por tanto,
el usuario ser uno de los destinatarios
self.correoRecibido -> includes(correo) and
correo.infoCorreo.destinatario -> includes(self)

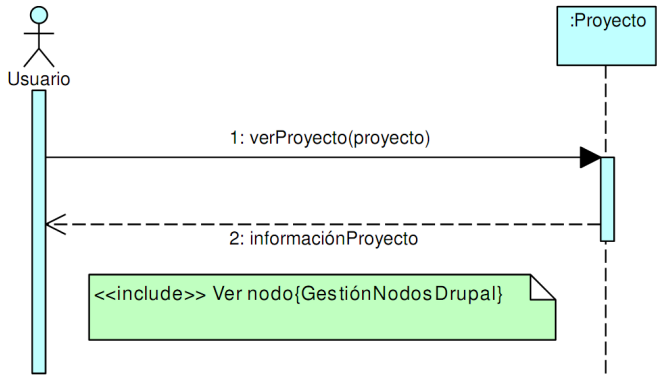
Salida:
Se recopila la información necesaria para enviar.

```
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{razón = razón,
        InfoExtra = 'Tipo contenido: Correo,
                IdentificadorCorreo: ' ->
                    concat(correo.identificador)}
```

3. Diagramas de secuencia de proyecto

Ver proyecto

Todos los usuarios tanto registrados como anónimos pueden ver la información pública de los proyectos. Esta información está compuesta por el título, roles o actividades solicitadas, tipo de proyecto, categoría y fecha de creación. Y dependiendo de la vista escogida, reducida o completa, incluye información extra. En caso de elegir la vista reducida verá la imagen y la descripción breve del proyecto. En cambio en la vista completa verá el vídeo o en su defecto la imagen y además la descripción completa.



Operación: verProyecto(proyecto:Proyecto, vista:String): TupleType(titulo:String, roles:Set(String), tipoProyecto: TipoProyecto, categoría: Categoría, descripción: String, fechaCreación: Fecha, imagen: Imagen, vídeo: Vídeo)

Descripción: Muestra la información del proyecto dependiendo del tipo de vista elegida
Precondiciones:

Debe existir el proyecto en el sistema

```
Proyecto.allInstances() -> exists(p| p = proyecto)
```

La vista será 'reducida' o 'completa'

```
vista = 'reducida or vista = 'completa'
```

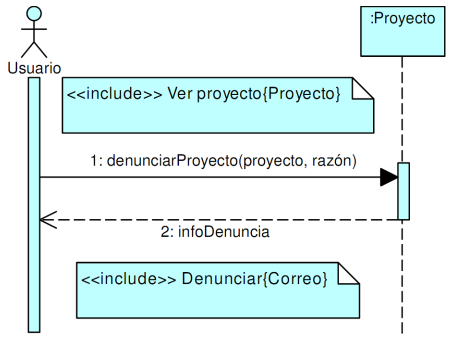
Salida:

En caso de elegir la vista reducida verá la imagen y la descripción breve del proyecto. En la vista completa verá el vídeo o en su defecto la imagen y la descripción completa. En ambas vistas verá el título, roles o actividades solicitadas, tipo de proyecto, categoría y fecha de creación.

```
let informaciónProyecto : TupleType(titulo: String, roles: Set(String), tipoProyecto: TipoProyecto, categoría: Categoría,
                                    descripción: String, fechaCreación: Fecha, imagen: Imagen, vídeo: Vídeo) in
  if vista = 'reducida' then
    estadísticas = Tuple{titulo = proyecto.título, roles = proyecto.roles, tipoProyecto = proyecto.tipoProyecto,
                         categoría = proyecto.categoría, descripción = proyecto.descripciónBreve,
                         fechaCreación = proyecto.fechaCreación, imagen: proyecto.imagen, vídeo -> oclIsUndefined()}
  elseif vista = 'completa' then
    let vídeo : Vídeo in
    let imagen : Imagen in
    if not proyecto.vídeo -> oclIsUndefined() then
      vídeo = proyecto.vídeo
      imagen -> oclIsUndefined()
    else
      vídeo -> oclIsUndefined()
      imagen = proyecto.imagen
    endif

    estadísticas = Tuple{titulo = proyecto.título, roles = proyecto.roles, tipoProyecto = proyecto.tipoProyecto,
                         categoría = proyecto.categoría, descripción = proyecto.descripciónCompleta,
                         fechaCreación = proyecto.fechaCreación, imagen = imagen, vídeo = vídeo}
  endif
```

Denunciar proyecto

El usuario denuncia un proyecto indicando una razón.



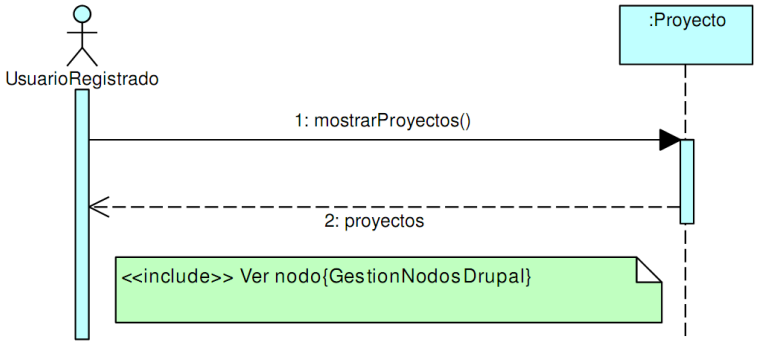
Operación: denunciarProyecto(proyecto: Proyecto, razón: String) :
TupleType(razón: String, infoExtra: String)
Descripción: Obtiene los datos necesarios para formalizar la denuncia
Precondiciones:
Debe existir el proyecto en el sistema
Proyecto.allInstances() -> exists(p | p = proyecto)

Salida:
Se recopila la información necesaria para enviar.

```
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{razón = razón,
        infoExtra = 'Tipo contenido: Proyecto, IdentificadorProyecto: '
            -> concat(proyecto.identificador)}
```

Mostrar proyectos

Muestra los proyectos en los que el usuario es administrador, miembro, invitado, candidato o rechazado según lo elegido por el usuario. Además también permite mostrar los proyectos destacados por el propio usuario.

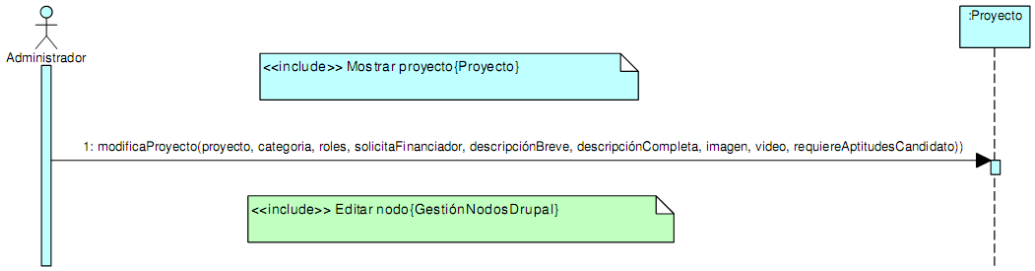


Operación: mostrarProyectos(tipo: String): Set(Proyecto)
Descripción: Muestra los proyectos por cierta categoría
Precondiciones: -
Salida:
Muestra un listado de los proyectos agrupados por membresía, expulsado o destacados.

```
let proyectos : Set(Proyecto) in
if tipo = 'Administrador' then
   proyectos = Proyecto.allInstances() -> select(p | p.creador = self)
elseif tipo = 'Miembro' then
   proyectos = Proyecto.allInstances() -> select(p | p.grupo.miembroAceptado.usuarioRegistrado -> includes(self))
elseif tipo = 'Invitado' then
  proyectos = Proyecto.allInstances() -> select(p | p.grupo.invitadoAceptado.usuarioRegistrado -> includes(self))
elseif tipo = 'Candidato' then
   proyectos = Proyecto.allInstances() -> select(p | p.grupo.candidatoAceptado.usuarioRegistrado -> includes(self))
elseif tipo = 'Rechazado' then
   Grupo.allInstances() -> select(g | g.miembroRechazado.usuarioRegistrado -> includes(self) or
                                      g.invitadoRechazado.usuarioRegistrado -> includes(self) or
                                      g.candidatoRechazado.usuarioRegistrado -> includes(self))
                                                -> iterate(g | proyectos -> includes(g.proyecto)))
elseif tipo = 'Destacado' then
   proyectos = Proyecto.allInstances()
                        -> select(p | p.grupo.userInfo.destacado and p.grupo.userInfo.usuarioRegistrado = self)
endif
```

Modificar proyecto

Modifica los datos públicos del proyecto, tales como categorías, roles, si necesita financiador, descripción breve y completa, imagen, vídeo e incluso si requiere la descripción de aptitudes del candidato en el momento de enviar una solicitud. El único dato de proyecto que no se puede modificar es el tipo de este.



Operación: modificaProyecto(proyecto: Proyecto, categoria: Categoría, roles: Set(String), solicitaFinanciador: Boolean, descripciónBreve: String, descripciónCompleta: String, imagen: Imagen, vídeo: Vídeo, requiereAptitudesCandidato: Boolean))
Descripción: Modifica datos públicos del proyecto, excepto el tipo de proyecto el cuál no se puede ya modificar.
Precondiciones:

El proyecto existe en el sistema

```
Proyecto.allInstances() -> exists(p | p = proyecto)
```

El usuario es el administrador del proyecto

```
proyecto.creador = self
```

Si la descripción breve no está vacía debe tener entre 1 y 300 caracteres

```
if not descripciónBreve -> oclIsUndefined() then
  descripciónBreve -> size() >= 1 and descripciónBreve -> size() <= 300
endif
```

Si la descripción completa no está vacía debe tener entre 1 y 150000 caracteres

```
if not descripciónCompleta -> oclIsUndefined() then
  descripciónCompleta -> size() >= 1 and descripciónCompleta -> size() <= 150000
endif
```

Postcondiciones:

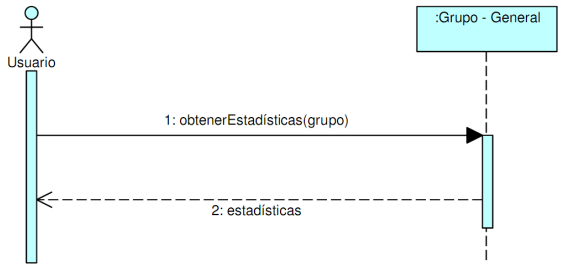
Modifica los datos introducidos por el usuario.

```
if not categoría -> oclIsUndefined() then
    proyecto.categoría = categoría
endif
if not roles -> oclIsUndefined() then
    proyecto.roles = roles
endif
if not solicitaFinanciador -> oclIsUndefined() then
    proyecto.solicitaFinanciador = solicitaFinanciador
endif
if not descripciónBreve -> oclIsUndefined() then
    proyecto.descripciónBreve = descripciónBreve
endif
if not descripciónCompleta -> oclIsUndefined() then
    proyecto.descripciónCompleta = descripciónCompleta
endif
if not imagen -> oclIsUndefined() then
    proyecto.imagen = imagen
endif
if not imagen -> oclIsUndefined() then
    proyecto.imagen = imagen
endif
if not requiereAptitudesCandidato -> oclIsUndefined() then
    proyecto.grupo.requiereAptitudesCandidato = requiereAptitudesCandidato
endif
```

4. Diagramas de secuencia general de grupo

Mostrar estadísticas y relación con grupo

Muestra datos del grupo como el porcentaje de objetivos cumplidos, la recaudación hasta el momento junto con su objetivo, indicando además si está activada o no. Además informa de la fecha prevista de finalización del proyecto y el estado de este, pudiendo ser abierto, cerrado, pendiente de eliminación o finalización, finalizado o eliminado. También informa de la relación existente entre el proyecto y el usuario, indicando la membresía, estado y rol o actividad si es que participa en él. Siendo las membresías posibles administrador, miembro, invitado o candidato y los estados aceptado, expulsado, pendiente de eliminación o aceptación, o simplemente no participante. Otros datos mostrados son si ha destacado el proyecto y cuantos otros usuarios lo han destacado también.



Operación: obtenerEstadísticas(grupo: Grupo) : TupleType(porcentajeCumplimiento: Integer, recaudado: Integer, objetivoRecaudación: Integer, recaudaciónActiva: Boolean, fechaPrevistaFinalización: Fecha, estadoProyecto: EstadoGrupo, membresía: TipoUsuario, rol: String, proyectoDestacado: Boolean, totalDestacadosProyecto: Integer, estadoUsuario: EstadoUsuario)

Descripción: Muestra estadísticas y relación con el proyecto

Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

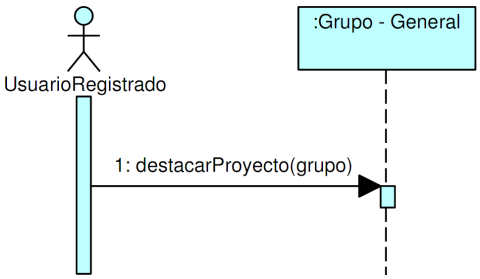
Salida:

Muestra estadísticas tales como lo recaudado, objetivo de recaudación, número de participantes y financiadores. Además indica la relación que hay con el proyecto, tales como la membresía y rol.

```
let estadísticas : TupleType(porcentajeCumplimiento: Integer, recaudado: Integer, objetivoRecaudación: Integer,
                             recaudaciónActiva: Boolean, fechaPrevistaFinalización: Fecha,
                                  estadoProyecto: EstadoGrupo, membresía: TipoUsuario, rol: String,
                             proyectoDestacado: Boolean, totalDestacadosProyecto: Integer,
                                  estadoUsuario: EstadoUsuario) in


 if self.oclIsTypeOf(UsuarioRegistrado) then
   let infoUsuario : InfoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
   estadísticas = Tuple{porcentajeCumplimento = grupo.porcentajeObjetivosCumplidos,
                              recaudado = grupo.recaudaciónTotal, objetivoRecaudación = grupo.objetivoRecaudación,
                            recaudaciónActiva = grupo.recaudaciónActiva,
                              fechaPrevistaFinalización = grupo.fechaObjetivo, estadoProyecto: grupo.estado,
                        membresía = infoUsuario.membresía, rol = infoUsuario.rol,
                              proyectoDestacado = InfoUsuario.destacaProyecto,
                              totalDestacadosProyecto = grupo.destacados, estadoUsuario = infoUsuario.estado}
 elseif self.oclIsTypeOf(UsuarioAnónimo) then
   estadísticas = Tuple{porcentajeCumplimento = grupo.porcentajeObjetivosCumplidos,
                            recaudado = grupo.recaudaciónTotal, objetivoRecaudación = grupo.objetivoRecaudación,
                              recaudaciónActiva = grupo.recaudaciónActiva,
                        fechaPrevistaFinalización = grupo.fechaObjetivo, estadoProyecto: grupo.estado,
                        membresía -> oclIsUndefined(), rol -> oclIsUndefined(),
                              proyectoDestacado -> oclIsUndefined(),
                        totalDestacadaosProyecto = oclIsUndefined(), estadoUsuario -> oclIsUndefined()}
 endif
```

Destacar proyecto

Destaca o deja de destacar un proyecto por parte del usuario.



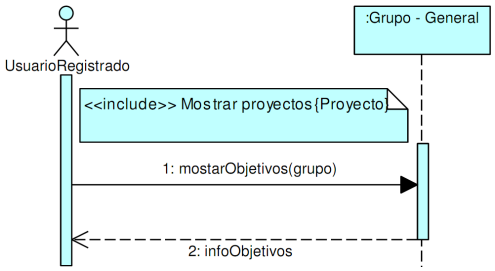
Operación: destacarProyecto(grupo: Grupo)
Descripción: Destaca o deja de destacar un proyecto.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Postcondiciones:
Destaca el proyecto si no lo había destacado o lo deja de destacar si ya estaba destacado por el usuario.

```
let grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
if infoUsuario.destacaProyecto then
  infoUsuario.destacaProyecto = false
  grupo.destacados = grupo.destacados - 1
else
   infoUsuario.destacaProyecto = true
   grupo.destacados = grupo.destacados + 1
endif
```

Ver objetivos

Muestra los objetivos añadidos por el administrador del proyecto. Estos objetivos ayudan no solo a los usuarios que visisten el proyecto a decidirse a participar sino que al propio administrador le facilita saber el progreso del propio proyecto y determinar un porcentaje de cumplimiento de estos objetivos.



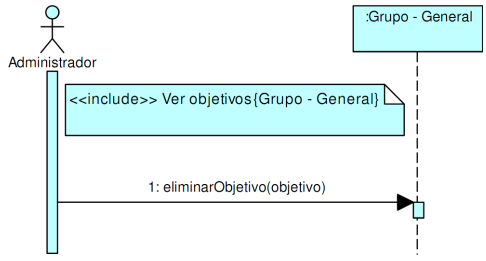
Operación: obtenerEstadísticas(grupo: Grupo) :
TupleType(porcentajeCumplimento: Integer,
objetivos: Set(Objetivo))
Descripción: Muestra estadísticas y relación con el proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Salida:
Genera un listado con los objetivos y sus porcentajes, además de porcentaje de cumplimiento alcanzado en total.

```
let infoObjetivos : TupleType(porcentajeCumplimento: Integer,
                              objetivos: Set(Objetivo) =
Tuple{porcentajoCumplimento  = grupo.porcentajeObjetivosCumplidos,
      objetivos = grupo.objetivo}
```

Eliminar objetivo

Elimina un objetivo añadido por el administrador al listado de objetivo del proyecto.



Operación: eliminarObjetivo(objetivo: Objetivo)
Descripción: Elimina el objetivo del sistema.
Precondiciones:
Debe existir el objetivo en el sistema
Objetivo.allInstances() -> exists(o | o = objetivo)

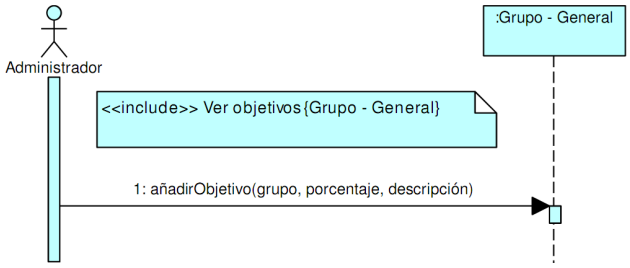
El objetivo es del mismo grupo del cuál el usuario es administrador
objetivo.grupo.creador.usuarioRegistrado = self

Postcondiciones:
Elimina el objetivo del sistema.

```
objetivo.grupo.objetivo -> excludes(objetivo) and
Objetivo.allInstances() -> excludes(objetivo)
```

Añadir objetivo

Añade un nuevo objetivo al grupo indicando una descripción del mismo y el porcentaje en el que se situaría dentro del progreso del proyecto. Teniendo en cuenta que no haya ningún otro objetivo del proyecto con el mismo porcentaje.



Operación: añadirObjetivo(grupo: Grupo, porcentaje: Integer, descripción: String)
Descripción: Añade un objetivo al proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Ninguno de los datos a introducir en el objetivo están vacíos
not (porcentaje -> oclIsUndefined() or descripción -> oclIsUndefined())

El porcentaje debe ser un entero entre 0 y 100
porcentaje >= 0 and porcentaje <= 100

El procentaje no puede ser igual a otro perteneciente a un objetivo del mismo grupo al que pertenece
grupo.objetivo -> select(o | o.porcentaje = porcentaje) -> isEmpty()

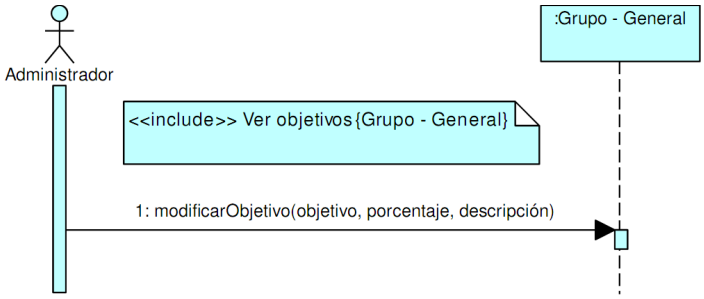
El administrador del grupo es el propio usuario
grupo.creador.usuarioRegistrado = self

Postcondiciones:
El objetivo es agregado al listado de objetivos del proyecto

let o : Objetivo -> oclIsUndefined() in
o.oclIsNew() and o.oclIsTypeOf(Objetivo) and o.identificador = Objetivo.allInstances().identificador -> max() + 1 and
o.porcentaje = porcentaje and o.descripción = descripción and grupo.objetivo -> includes(o)

Modificar objetivo

Modifica la descripción y/o el porcentaje del objetivo.



Operación: modificarObjetivo(objetivo: Objetivo, porcentaje: Integer, descripción: String)
Descripción: Modifica los datos de un objetivo
Precondiciones:

Debe existir el objetivo en el sistema
```
Objetivo.allInstances() -> exists(o | o = objetivo)
```

El objetivo es del grupo del cuál el usuario es administrador
```
objetivo.grupo.creador.usuarioRegistrado = self
```

El porcentaje debe ser un entero entre 0 y 100
```
porcentaje >= 0 and porcentaje <= 100
```

El procentaje no puede ser igual a otro perteneciente a un objetivo del mismo grupo al que pertenece
```
grupo.objetivo -> select(o | o.porcentaje = porcentaje) -> isEmpty()
```

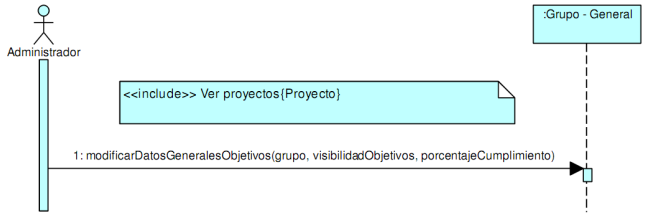
Postcondiciones:

Modifica los datos del objetivo que haya indicado el administrador.

```
if not porcentaje -> oclIsUndefined() then
  objetivo.porcentaje = porcentaje
endif

if not descripción -> oclIsUndefined() then
  objetivo.descripción = descripción
endif
```

Modificar datos generales de objetivos

Modifica la visibilidad de los objetivos que puede para todos los usuarios registrados, para solo participantes del proyecto(administrador, invitados, miembros y candidatos), para participantes menos los canidatos o solo para administrador y miembro. Además permite modificar el porcentaje de cumplimiento del proyecto.



Operación: modificarDatosGeneralesObjetivos(grupo: Grupo, visibilidadObjetivos: VisibilidadObjetivos, porcentajeCumplimiento: Integer)
Descripción: Modifica la visibilidad de objetivos y el porcentaje total de cumplimiento
Precondiciones:

Debe existir el grupo en el sistema
```
Grupo.allInstances() -> exists(g | g = grupo)
```

El administrador lo es del mismo grupo al que pertenece el objetivo.
```
objetivo.grupo.creador.usuarioRegistrado = self
```

El porcentaje de cumplimiento si es indicado debe ser un entero entre 0 y 100
```
porcentaje -> oclIsUndefined() or (not porcentaje -> oclIsUndefined() and porcentaje >=  0 and porcentaje <= 100))
```

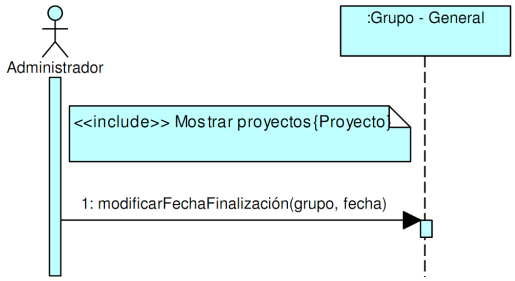
Postcondiciones:

Los datos de visibilidad y/o porcentaje total de cumplimiento son modificados

```
if not visibilidadObjetivos -> oclIsUndefined() then
  grupo.visibilidadObjetivos = visibilidadObjetivos
endif

if not porcentajeCumplimiento -> oclIsUndefined() then
  grupo.porcentajeObjetivosCumplidos = porcentajeCumplimiento
endif
```

Modificar fecha finalización del proyecto

Modifica la fecha prevista de finalización que es un dato meramente orientativo y que no influirá en ningún caso en el funcionamiento del resto de funcionalidades.



Operación: modificarFechaFinalización(grupo: Grupo, fecha: Fecha)
Descripción: Modifica la fecha prevista de finalización
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

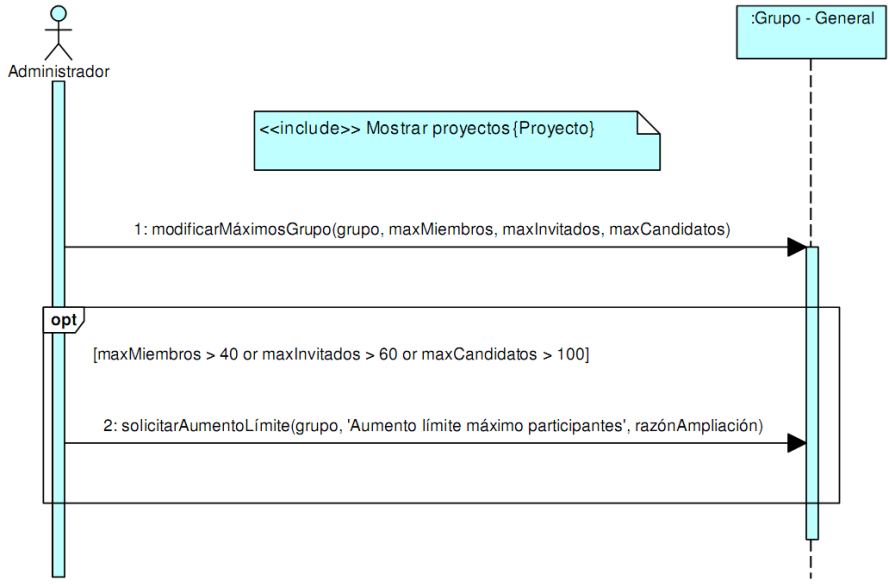
El administrador del grupo es el propio usuario
grupo.creador.usuarioRegistrado = self

Postcondiciones:
La fecha prevista de finalización queda modificada

```
if fecha -> oclIsUndefined() then
   grupo.fechaObjetivo -> oclIsUndefined()
else
   grupo.fechaObjetivo = fecha
endif
```

Modificar máximos grupo

Modifica el número de participantes por cada membresía máximos que pueden estar asociados a la vez en un proyecto. Esta funcionalidad permite tener control en el número de participantes para no sobrepasar un límite. Todo máximo que aquí se modifique tendrá que ser un número como mínimo igual al número actual de participantes para la membresía a modificar y como máximo el límite global del sistema para esa membresía. En caso de tener que superar ese límite global, el administrador del proyecto deberá contactar con el administrador del sistema.



Operación: modificarMáximosGrupo(grupo: Grupo, maxMiembros: Integer, maxInvitados: Integer, maxCandidatos: Integer)
Descripción: Modifica los máximos de candidatos, invitados y/o miembros del proyectos que pueden participar a la vez
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El administrador del grupo es el propio usuario

```
grupo.creador.usuarioRegistrado = self
```

El máximo de miembros debe ser un entero entre el total de miembros actual y 40

```
maxMiembros >= grupo.totalMiembros and maxMiembros <= 40
```

El máximo de invitados debe ser un entero entre el total de invitados actual y 60

```
maxInvitados >= grupo.totalInvitados and maxInvitados <= 60
```

El máximo de candidatos debe ser un entero entre el total de candidatos actual y 100

```
maxCandidatos >= grupo.totalCandidatos and maxCandidatos <= 100
```

Postcondiciones:

Los máximos de candidatos, invitados y/o miembros del grupo quedan modificados

```
if not maxMiembros -> oclIsUndefined() then
    grupo.maxMiembros = maxMiembros
endif
if not maxInvitados -> oclIsUndefined() then
     grupo.maxInvitados = maxInvitados
 endif
if not maxCandidatos -> oclIsUndefined() then
     grupo.maxCandidatos = maxCandidatos
endif
```

Operación: solicitarAumentoLímite(grupo: Grupo, asunto: String, razónAmpliación: String)
Descripción: Envia un mail al administrador del sistema solicitando ampliación del límite.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El administrador del grupo es el propio usuario

```
grupo.creador.usuarioRegistrado = self
```

Postcondiciones:

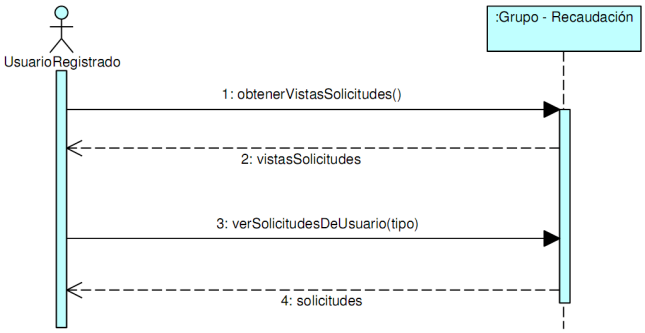
Se solicita la ampliación de los límites de máximos del grupo al administrador del sistema.

```
let mensaje : MensajeAdminSistema -> oclIsUndefined() in
mensaje.oclIsNew() and mensaje.oclIsTypeOf(MensajeAdminSistema) and mensaje.asunto = asunto and
mensaje.usuario = self and mensaje.texto = razónAmpliación and mensaje.destacado = false
infoExtra = "proyecto con id: ".concat(grupo.identificador.toString()) and
mensaje.identificador = Mensaje.allInstances().identificador -> max() + 1 and mensaje.leido = false
```

5. Diagramas de secuencia asociación a grupo

Mostrar solicitudes enviadas de usuario

Muestra el listado de solicitudes enviadas a proyectos por el usuario, clasificadas por vistas. Estas vistas muestran las solicitudes pendientes, aceptadas, rechazadas o todas.



Operación: obtenerVistasSolicitudes() : Set(String)
Descripción: Muestra el listado de vistas de solicitudes disponibles
Precondiciones: -
Salida:

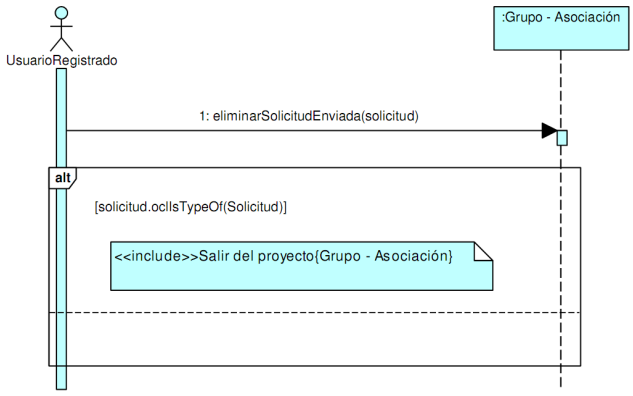
```
let vistasSolicitudes : Set(String) -> oclIsUndefined() in
vistasSolicitudes -> includes('Todas') -> includes('Pendientes')
                  -> includes('Aceptadas') -> includes('Rechazadas')
```

Operación: verSolicitudesEnviadasDeUsuario(tipo: String): Set(TupleType(proyecto: Proyecto, solicitud: Solicitud))
Descripción: Comprueba si la solicitud está en espera, aceptada o rechazada.
Precondiciones: -
Salida:
Genera el listado de solicitudes pendientes, aprobadas y rechazadas

```
let solicitudes : Set(TupleType(proyecto: Proyecto, solicitud: Solicitud)) in
if tipo = 'Todas' then
  self.infoUsuario.oclAsTypeOf(Candidato).solicitud
                -> iterate(s | solicitudes -> includes(Tuple(proyecto = s.grupo.proyecto, solicitud = s)))
  self.infoUsuario.oclAsTypeOf(Candidato).solicitudRechazada
                -> iterate(s | solicitudes -> includes(Tuple(proyecto = s.grupo.proyecto, solicitud = s)))
  self.infoUsuario.oclAsTypeOf(Invitado).solicitudAprobada
                -> iterate(s | solicitudes -> includes(Tuple(proyecto = s.grupo.proyecto, solicitud = s)))
elseif tipo = 'Pendientes' then
  self.infoUsuario.oclAsTypeOf(Candidato).solicitud
                -> iterate(s | solicitudes -> includes(Tuple(proyecto = s.grupo.proyecto, solicitud = s)))
elseif tipo = 'Rechazadas' then
  self.infoUsuario.oclAsTypeOf(Candidato).solicitudRechazada
                -> iterate(s | solicitudes -> includes(Tuple(proyecto = s.grupo.proyecto, solicitud = s)))
elseif tipo = 'Aceptadas' then
  self.infoUsuario.oclAsTypeOf(Invitado).solicitudAprobada
                -> iterate(s | solicitudes -> includes(Tuple(proyecto = s.grupo.proyecto, solicitud = s)))
endif
```

Eliminar solicitud

Elimina una solicitud enviada que solo en el caso de estar pendiente de aceptación provoca que el usuario salga del proyecto.



Operación: eliminarSolicitudEnviada(solicitud: Solicitud)
Descripción: Elimina una solicitud enviada por él mismo a algún proyecto
Precondiciones:
Debe existir la solicitud en el sistema
Solicitud.allInstances() -> includes(solicitud) or SolicitudRechazada.allInstances() -> includes(solicitud) or
SolicitudAceptada.allInstances() -> includes(solicitud)

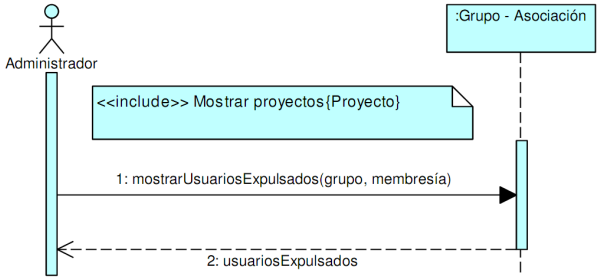
El candidato lo es del propio grupo al que pertenece la solicitud
solicitud.grupo.candidatoAceptado.usuarioRegistrado -> includes(self) or
solicitud.grupo.candidatoRechazado.usuarioRegistrado -> includes(self) or
solicitud.grupo.invitadoAceptado.usuarioRegistrado -> includes(self)

Postcondiciones:
La solicitud es eliminada del sistema.

```
if solicitud.oclIsTypeOf(Solicitud) then
   solicitud.grupo -> oclIsUndefined() and solicitud.candidato ->oclIsUndefined() and
   Solicitud.allInstances() -> excludes(solicitud)
elseif solicitud.oclIsTypeOf(SolicitudRechazada) then
   solicitud.grupo -> oclIsUndefined() and solicitud.candidato ->oclIsUndefined() and
   SolicitudRechazada.allInstances() -> excludes(solicitud)
elseif solicitud.oclIsTypeOf(SolicitudAceptada) then
   solicitud.grupo -> oclIsUndefined() and solicitud.invitado ->oclIsUndefined() and
   SolicitudAceptada.allInstances() -> excludes(solicitud)
endif
```

Ver usuarios expulsados del grupo

Muestra los usuarios expulsados del grupo, dependiendo de la membresía escogida para visualizar.



Operación: mostrarUsuariosExpulsados(grupo: Grupo, membresía: TipoUsuario) : Set(UsuarioRegistrado)
Descripción: Muestra el listado de usuarios expulsados del grupo
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El administrador lo es del propio grupo
grupo.creador.usuarioRegistado = self

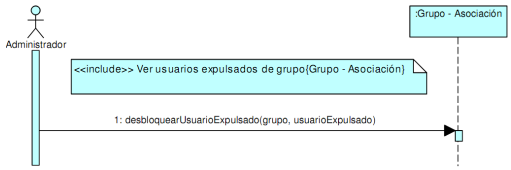
La membresía será 'Miembro', 'Invitado' o 'Candidato'
membresía = TipoUsuario::Miembro or membresía = TipoUsuario::Invitado or membresía = TipoUsuario::Candidato

Salida:
Genera el listado de usuarios expulsados del grupo

```
let usuariosExpulsados : Set(UsuarioRegistrado) in
if membresía = TipoUsuario::Miembro then
   usuariosExpulsados = grupo.miembroExpulsado.usuarioRegistrado
elseif membresía = TipoUsuario::Invitado then
   usuariosExpulsados = grupo.invitadoExpulsado.usuarioRegistrado
elseif membresía = TipoUsuario::Candidato then
   usuariosExpulsados = grupo.candidatoExpulsado.usuarioRegistrado
endif
```

Desbloquear usuario expulsado del grupo

Desbloquea la expulsión de un usuario del proyecto para que en el momento que lo precise pueda volver a enviar una solicitud de entrada sin tener que esperar 4 meses. No confundir este bloqueo con el que puede hacer cada usuario a otro de forma individual. En caso de ser un candidato expulsado, de existir aun la solicitud y cumplirse las precondiciones para enviar la solicitud, automáticamente esta solicitud pasa de rechazada a pendiente de nuevo y el usuario vuelve a ser un candidato aceptado a la espera que acepten su solicitud.



Operación: desbloquearUsuarioExpulsado(grupo: Grupo, usuarioExpulsado: UsuarioRegistrado)
Descripción: Desbloquea un usuario expulsado del proyecto. De esta forma no ha de esperar los 4 meses desde la expulsión para volver a
solicitar la entrada en el grupo.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

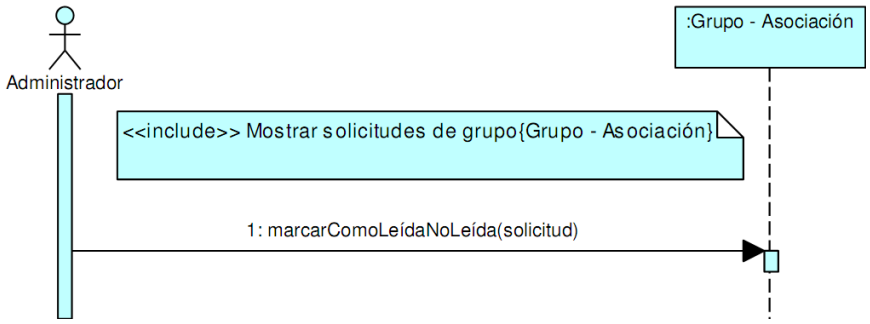
El administrador lo es del propio grupo
grupo.creador.usuarioRegistrado = self

El usuarioExpulsado está expulsado del grupo
grupo.candidatoExpulsado.usuarioRegistrado -> includes(usuarioExpulsado) or
grupo.invitadoExpulsado.usuarioRegistrado -> includes(usuarioExpulsado) or
grupo.miembroExpulsado.usuarioRegistrado -> includes(usuarioExpulsado)

Postcondiciones:
El usuario queda desbloqueado y ya puede solicitar de nuevo entrar en el proyecto.
En caso de ser un candidato bloqueado y existir aún la solicitud, vuelve a quedar aceptado como candidato.

```
let infoUsuario : InfoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
infoUsuario.estado = EstadoUsuario::Aceptado and infoUsuario.estado = TipoUsuario::NoParticipa

if grupo.candidatoExpulsado.usuarioRegistrado -> includes(usuarioExpulsado) then
    let solicitudRechazada : SolicitudRechazada =
                grupo.solicitudRechazada -> select(s | s.candidato.usuarioRegistrado = usuarioExpulsado) in
    if not solicitudRechazada -> oclIsUndefined() and grupo.totalCandidatos < grupo.maxCandidatos then
      grupo.candidatoExpulsado -> excludes(infoUsuario) and grupo.candidatoAceptado -> includes(infoUsuario) and
      solicitudRechazada.oclIsTypeOf(Solicitud) and grupo.totalCandidatos = grupo.totalCandidatos + 1 and
      infoUsuario.rol = solicitud.rol and infoUsuario.membresía = TipoUsuario::Candidato and
        infoUsuario.fecha = fechaActual()
        if infoUsuario.pagoPendiente <> 0 then
         let pagoPendiente : PagoPendiente =
                          infoUsuario.usuarioRegistrado.pagoPendiente -> select(p | p.grupo = grupo) in
        usuarioRegistrado.pagoPendiente -> excludes(pagoPendiente) and
          PagoPendiente.allInstances() -> excludes(pagoPendiente) and solicitud.inversión = pagoPendiente.cantidad
      else
        infoUsuario.pagoPendiente = solicitud.inversión
      endif
    else
      grupo.candidatoExpulsado -> excludes(infoUsuario) and
      infoUsuario.membresía = TipoUsuario::NoParticipa and infoUsuario.fecha -> oclIsUndefined()
    endif
  elseif grupo.invitadoExpulsado.usuarioRegistrado -> includes(usuarioExpulsado) then
    grupo.candidatoExpulsado -> excludes(infoUsuario) and
    infoUsuario.membresía = TipoUsuario::NoParticipa and infoUsuario.fecha -> oclIsUndefined()
  elseif grupo.miembroExpulsado.usuarioRegistrado -> includes(usuarioExpulsado) then
    grupo.miembroExpulsado -> excludes(infoUsuario) and
    infoUsuario.membresía = TipoUsuario::NoParticipa and infoUsuario.fecha -> oclIsUndefined()
  endif
```

Marcar solicitud como leída o no leída



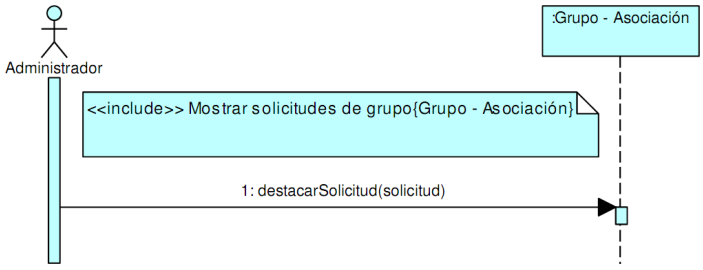
Operación: marcarComoLeídaNoLeída(solicitud: Solicitud)
Descripción: Marca una solicitud como leída o no leída.
Precondiciones:
Debe existir la solicitud en el sistema
Solicitud.allInstances() -> exists(s | s = solicitud)

El administrador lo es del propio grupo al que pertenece la solicitud
solicitud.grupo.creador.usuarioRegistrado = self

Postcondiciones:
Marca la solicitud como leída si no lo estaba o la marca como no leída si lo estaba

```
if solicitud.leida then
    solicitud.leida = false
else
    solicitud.leida= true
endif
```

Destacar solicitud



Operación: destacarSolicitud(solicitud: Solicitud)
Descripción: Destaca o deja de destacar una solicitud
Precondiciones:
Debe existir la solicitud en el sistema
Solicitud.allInstances() -> exists(s | s = solicitud)

El administrador lo es del propio grupo al que pertenece la solicitud
solicitud.grupo.creador.usuarioRegistrado = self

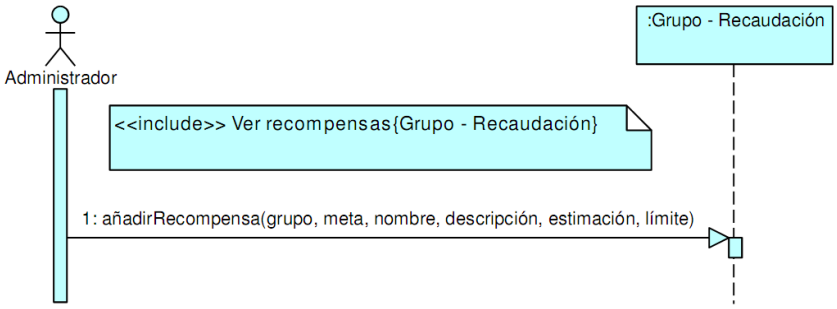
Postcondiciones:
Destaca la solicitud si no estaba destacada o deja de destacarla si ya estaba destacada

```
if solicitud.destacada then
    solicitud.destacada = false
else
    solicitud.destacada = true
endif
```

6. Diagramas de secuencia financiación de grupo

Añadir recompensa

Añade una recompensa nueva al grupo. Esta recompensa se compone de un nombre, una meta, siendo la cantidad mínima de la aportación económica para poder solicitarla, una descripción, una estimación de semanas que tardaría en estar preparada para enviar una vez completado el pago y un límite en el número de recompensas solicitadas.



Operación:añadirRecompensa(grupo: Grupo, meta: Integer, nombre: String, descripción: String, estimación: Integer, límite: Integer)
Descripción: Añade una nueva recompensa al grupo
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

La recaudación y el sistema de recompensas del grupo deben estar activados
grupo.recaudaciónActiva = true and grupo.sistemaRecompensasActivo = true

Ninguno de los datos a introducir en la recompensa estás vacíos
not (meta -> oclIsUndefined() or nombre -> oclIsUndefined() or estimación -> oclIsUndefined() or descripción -> oclIsUndefined() or estimación -> oclIsUndefined() or límite -> oclIsUndefined())

La meta y la estimación deben ser mayores o igual que 1.
meta >= 1 and estimación >= 1

El nombre de la recompensa debe tener entre 1 y 60 caracteres
nombre -> size() >= 1 and nombre -> size() <= 60

La descripción de la recompensa debe tener entre 1 y 200 caracteres
descripción -> size() >= 1 and descripción -> size() <= 200

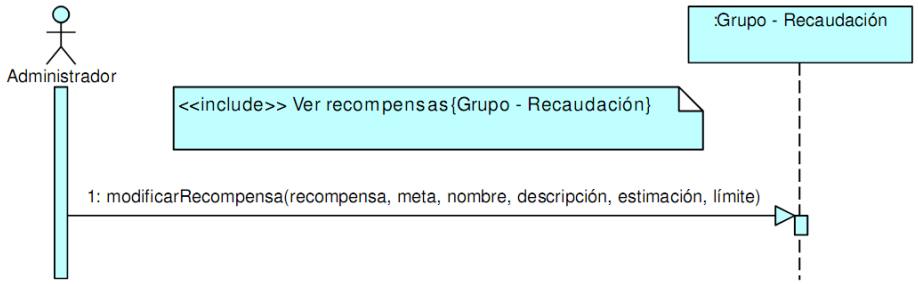
La meta no puede ser igual a otra perteneciente a una recompensa del mismo grupo al que pertenece
grupo.recompensa -> select(r | r.meta = meta) -> isEmpty()

El administrador del grupo es el propio usuario
grupo.creador.usuarioRegistrado = self

Postcondiciones:
Crea una nueva recompensa y la asocia al grupo
let r : recompensa -> oclIsUndefined() in
r.oclIsNew() and r.oclIsTypeOf(Recompensa) and r.nombre = nombre and r.descripción = descripción and r.identificador = Recompensa.allInstances().identificador -> max() + 1 and r.meta = meta and r.estimación = estimación and r.límite = límite and grupo.recompensa -> includes(r)

Modificar recompensa

Modifica datos de la recompensa. Estos datos no afectan a las recompensas solicitadas, por tanto, una vez solicitadas aunque la meta suba o baje la recompensa solicitada deberá ser enviada igualmente.



Operación: modificarRecompensa(recompensa: Recompensa, meta: Integer, nombre: String, descripción: String, estimación: Integer, límite: Integer)

Descripción: Modifica los datos de la recompensa.

Precondiciones:

Debe existir la recompensa en el sistema

```
Recompensa.allInstances() -> exists(r | r = recompensa)
```

La recompensa es del grupo del cuál el usuario es administrador

```
recompensa.grupo.creador.usuarioRegistrado = self
```

La recaudación y el sistema de recompensas del grupo al que pertenece la recompensa deben estar activados

```
recompensa.grupo.recaudaciónActiva = true and recompensa.grupo.sistemaRecompensasActivo = true
```

El nombre de la recompensa, si es introducido, debe tener entre 1 y 60 caracteres

```
nombre -> oclIsUndefined() or (not nombre -> oclIsUndefined() and nombre -> size() >= 1 and nombre -> size() <= 60)
```

La descripción de la recompensa, si es introducida, debe tener entre 1 y 200 caracteres

```
descripción -> oclIsUndefined() or
(not descripción -> oclIsUndefined() and descripción -> size() >= 1 and descripción -> size() <= 200)
```

La meta y la estimación, si son introducidas, deben ser mayores o igual que 1.

```
(meta -> oclIsUndefined() or (not meta -> oclIsUndefined() and meta >= 1)) and
(estimación -> oclIsUndefined() or (not estimación -> oclIsUndefined() and estimación >= 1))
```

La meta no puede ser igual a otra perteneciente a una recompensa del mismo grupo al que pertenece

```
recompensa.grupo.recompensa -> select(r | r.meta = meta) -> isEmpty()
```

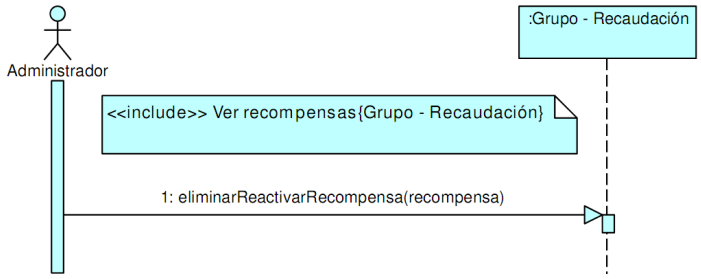
Postcondiciones:

Modifica los datos de la recompensa que el administrador haya indicado

```
if not meta -> oclIsUndefined() then
   recompensa.meta = meta
endif
if not nombre -> oclIsUndefined() then
   recompensa.nombre = nombre
endif
if not descripción -> oclIsUndefined() then
     recompensa.descripción = descripción
endif
if not estimación -> oclIsUndefined() then
     recompensa.estimación = estimación
endif
if not limite -> oclIsUndefined() then
     recompensa.límite = límite
endif
```

Eliminar/Reactivar recompensa

Elimina una recompensa activada. En caso de que tenga recompensas solicitadas, no sería eliminada del sistema pero si que no sería ofrecida a los usuarios que vayan a financiar el proyecto. Por tanto, hasta que no tenga ninguna recompensa solicitada asociada podrá reactivarse cuando el administrador lo necesite.



Operación: eliminarRecompensa(recompensa: Recompensa)
Descripción: Elimina una recompensa de un proyecto si está activada o la reactiva si estaba desactivada.
Precondiciones:

Debe existir la recompensa en el sistema

```
Recompensa.allInstances() -> exists(r | r = recompensa)
```

La recaudación y el sistema de recompensas del grupo al que pertenece la recompensa deben estar activados

```
recompensa.grupo.recaudaciónActiva = true and recompensa.grupo.sistemaRecompensasActivo = true
```

La recompensa es del mismo grupo del cual el usuario es administrador

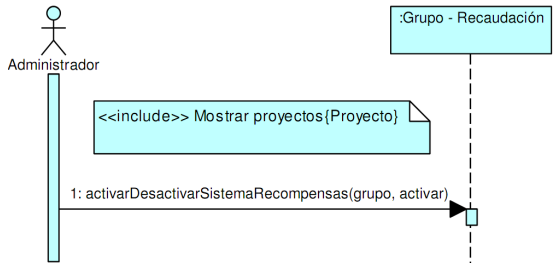
```
recompensa.grupo.creador.usuarioRegistrado = self
```

Postcondiciones:

Elimina la recompensa del sistema en caso de no haber recompensas solicitadas para esta. En caso de haberlas, la recompensa es desactivada. Si la recompensa ya estaba desactivada entonces la reactiva.

```
if recompensa.activada then
   if recompensa.solicitadas = 0 then
     recompensa.grupo.recompensa -> excludes(recompensa) and Recompensa.allInstances -> excludes(recompensa)
   else
     recompensa.activada = false
   endif
else
   recompensa.activada = true
endif
```

Activar/Desactivar sistema de recompensas



Operación: activarDesactivarSistemaRecompensas(grupo: Grupo, activar: Boolean)
Descripción: Activa o desactiva el sistema de recompensas
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

La recaudación del grupo está activada

```
grupo.recaudaciónActiva = true
```

El usuario participa como administrador del proyecto

```
grupo.creador.usuarioRegistrado = self
```

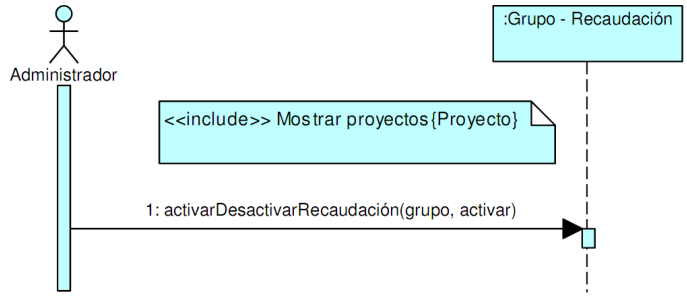
Postcondiciones:

Activa o desactiva el sistema de recompensas

```
grupo.sistemaRecompensasActivo = activar
```

Activar/Desactivar recaudación

Si se desactiva la recaudación elimina los pagos pendientes, menos de los miembros financiadores. Los miembros no financiadores que estaban pendientes de realizar un pago para formar parte del proyecto son convertidos en miembros de pleno derecho. El pago pendiente queda anulado también para resto de usuarios con pagos pendientes y que no tengan el rol de financiador.



Operación: activarDesactivarRecaudación(grupo: Grupo, activar: Boolean)
Descripción: Activa o desactiva la recaudación
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El usuario participa como administrador del proyecto.

```
grupo.creador.usuarioRegistrado = self
```

Postcondiciones:

Activa o desactiva la recaudación

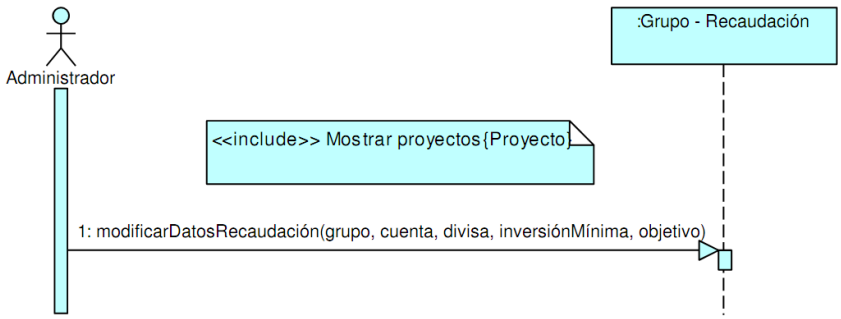
```
grupo.recaudaciónActiva = activar
```

Si se desactiva la recaudación elimina los pagos pendientes, menos de los miembros financiadores.
Los miembros no financiadores que estaban pendientes de realizar un pago para formar parte del proyecto
son convertidos en miembros de pleno derecho. El pago pendiente queda anulado también para resto de usuarios con
pagos pendientes y que no tengan el rol de financiador.

```
if not activar then
   grupo.PagoPendiente -> excludesAll(grupo.PagoPendiente
                                                    ->select(p | p.tipo <> TipoPagoPendiente::PagoDeFinanciador))
   grupo.infoUsuario -> select(u | u.rol <> 'financiador' and u.membresia = TipoUsuario::Miembro and
                                          u.estado = EstadoUsuario::PendienteAceptación)
                              -> forAll(u | u.pagoPendiente = 0 and u.estado = EstadoUsuario::Aceptado) and
   grupo.infoUsuario -> select(u | u.rol <> 'financiador' and pagoPendiente <> 0) -> forAll(u | u.pagoPendiente = 0) and
endif
```

Modificar datos recaudación

Modifica los datos tales como la cuenta de destino de la recaudación, la divisa que utiliza, solo siendo válidas las que utiliza el gestor de transacciones, la inversión mínima para usuarios con rol de financiador y la cifra objetivo.



Operación: modificarCuenta(grupo: Grupo, cuenta: String, divisa: String, inversiónMínima: Integer, objetivo: Integer)
Descripción: Modifica los datos de recaudación introducidos por el administrador de un proyecto concreto.
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

El usuario participa como administrador en el proyecto del cuál se modificarán los datos de recaudación.

```
grupo.creador.usuarioRegistrado = self
```

La recaudación de grupo está activada

```
grupo.recaudaciónActiva = true
```

En caso de indicar la cuenta debe tener entre 1 y 64 caracteres

```
if not cuenta-> oclIsUndefined() then
   cuenta -> size() >= 1 and cuenta -> size() <= 64
endif
```

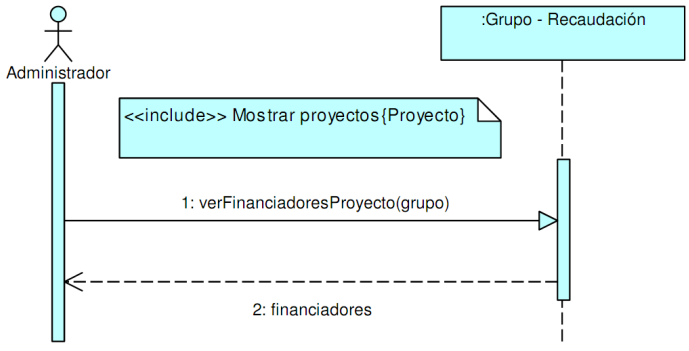
Postcondiciones:

Modifica los atributos introducidos por el administrador

```
if not cuenta-> oclIsUndefined() then
  grupo.cuenta = cuenta
endif
if not divisa -> oclIsUndefined() then
  grupo.divisa = divisa
endif
if not inversiónMínima -> oclIsUndefined() then
  grupo.inversiónMínima = inversiónMínima
endif
if not objetivo -> oclIsUndefined() then
  grupo.objetivoRecaudación = objetivo
endif
```

Ver financiadores de proyecto

Muestra la lista de financiadores de un proyecto en concreto. Junto con cada financiación aparece el financiador, la fecha, cantidad, membresía y recompensa, si es que la solicitó.



Operación: verFinanciadoresProyecto(grupo: Grupo): Set(TupleType(usuario: Usuario, cantidad: Integer, fecha: Fecha, membresía: TipoUsuario, recompensa: String))
Descripción: Genera un listado de información sobre quién ha financiado el proyecto.
Precondiciones:
Debe existir la recompensa en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

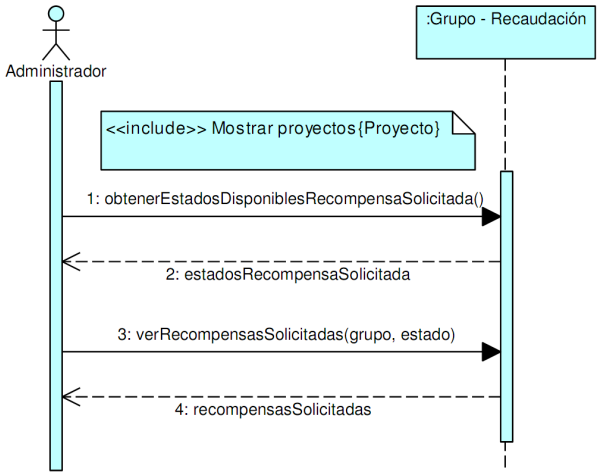
El administrador del grupo es el propio usuario
grupo.creador.usuarioRegistrado = self

Salida:
Retorna una colección de información sobre los financiadores del proyecto.

```
let financiadores: Set( TupleType(usuario: Usuario, cantidad: Integer, fecha: Fecha, membresía: TipoUsuario,
                                  recompensa: String)) in
grupo.pagoCompletado
      -> forAll(p | financiadores -> union(Tuple{usuario = p.usuarioRegistrado, cantidad = p.cantidad, fecha = p.fecha,
                                  membresía = grupo.infoUsuario -> select(i | i.usuarioRegistrado = p.usuario).membresía,
                                  recompensa = p.recompensaSolicitada.recompensa.nombre}))
```

Ver recompensas solicitadas de grupo

Muestra el listado de recompensas solicitadas de un proyecto clasificadas por el estado de la recompensa.



Operación: obtenerEstadosDisponiblesRecompensasSolicitadas(): Set(EstadoRecompensa)
Descripción: Muestra el listado de vistas disponibles en las que están clasificadas las recompensas solicitadas.
Precondiciones: -
Salida:
Genera el listado de estados disponibles para las recompensas solicitadas.

```
let estadosRecompensaSolicitada : Set(EstadoRecompensa) = EstadoRecompensa.allInstances()
```

Operación: verRecompensasSolicitadas(grupo: Grupo, estado: EstadoRecompensa): Set(RecompensaSolicitada)
Descripción: Genera el listado de de recompensas solicitadas del proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

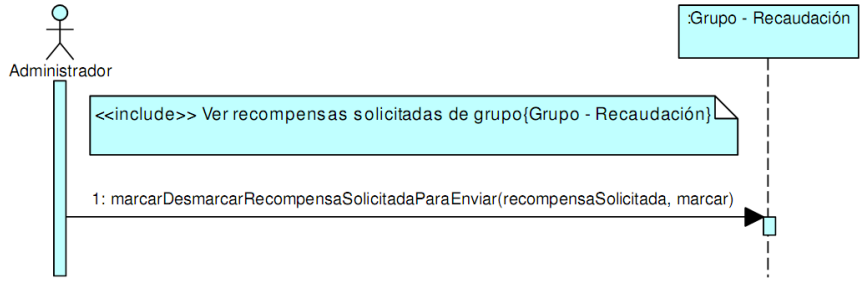
El usuario participa como administrador en el proyecto
grupo.creador.usuarioRegistrado = self

Salida:
Genera el listado de de recompensas solicitadas del proyecto dependiendo del estado indicado

```
let recompensasSolicitadas : Set(RecompensaSolicitada) -> oclIsUndefined() in
recompensasSolicitadas = grupo.recompensa.recompensasSolicitadas -> select(r | r.estado = estado)
```

Ver recompensas solicitadas de grupo

Marcar o desmarca una recompensa solicitada como preparada para enviar. Esta opción está disponible siempre y cuando el usuario receptor de la recompensa haya completado el pago. Con esta funcionalidad el usuario comprueba que en un lapso de pocos días podría recibir la recompensa. Una vez recibida el usuario solo tendría que marcarla como recibida, siendo esto otra funcionalidad explicada más adelante. La misma funcionalidad puede ser utilizada para retractarse e indicar que aun no está preparada.



Operación: marcarDesmarcarRecompensaSolicitadaParaEnviar(recompensaSolicitada: RecompensaSolicitada, marcar: Boolean)
Descripción: Marca o desmarca una recompensa solicitada como preparada para enviar.
Precondiciones:
Debe existir la recompensa solicitada en el sistema
RecompensaSolicitada.allInstances() -> exists(r | r = recompensaSolicitada)

El usuario participa como administrador en el proyecto al que pertenece la recompensa solicitada
recompensaSolicitada.recompensa.grupo.creador.usuarioRegistrado = self

El pago pendiente de la recompensa solicitada está completado
recompensaSolicitada.transacción.oclIsTypeOf(PagoCompletado)

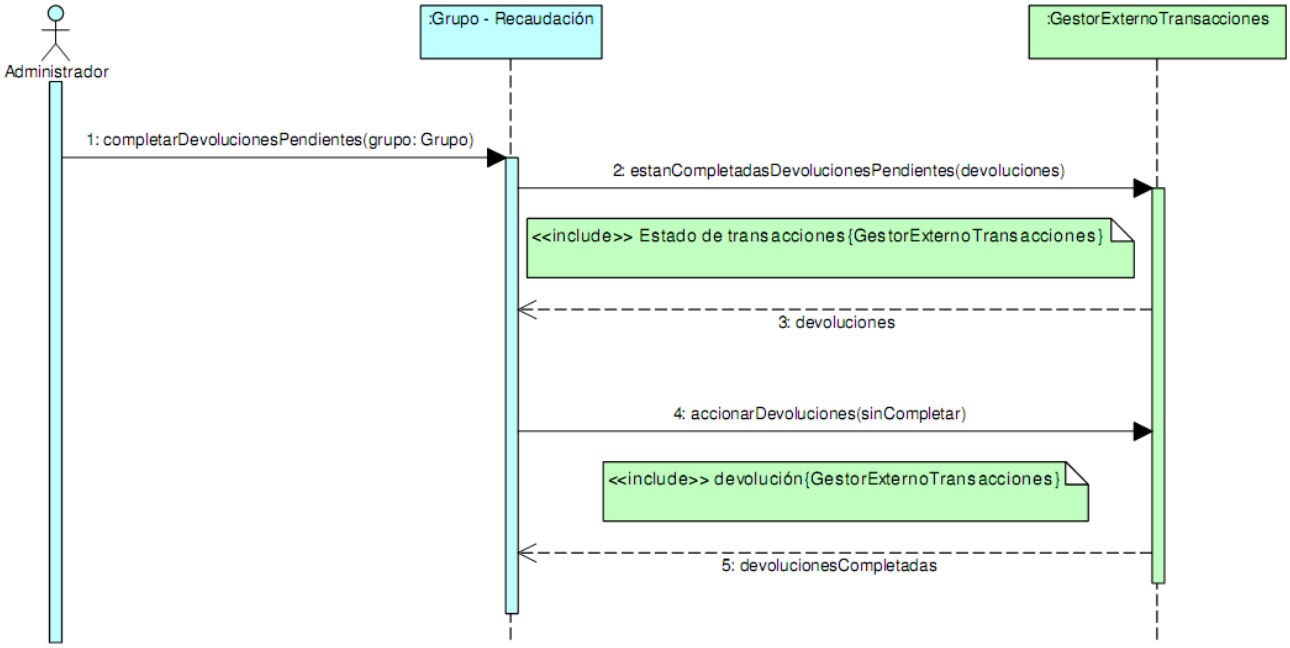
Postcondiciones:
Marca o desmarca una recompensa solicitada como peparada para enviar según lo indicado por el administrador.

```
if marcar then
    recompensaSolicitada.estado = EstadoRecompensa::PendienteDeEnvío
else
    recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado
endif
```

Completar devoluciones pendientes

El sistema transmite los datos de las devoluciones al gestor de transacciones. Esta acción no puede ser automatizada ya que requiere permiso expreso del propio administrador del proyecto y este permiso es concendido solo por un lapso de tiempo establecido por el gestor de transacciones. Primero comprueba que por un intento anterior no hayan sido ya completadas y de las que quedan por completar son comunicados de nuevo los datos de estas para resolverlas. Cada devolución completada que tuviese una recompensa solicitada adjunta y por tanto, con estado pendiente de devolución de pago, será eliminada del sistema y actualizados los datos de recompensa, tales como el número de solicitadas. En caso de que la recompensa estuviese desactivada, y no hubiesen ya recompensas solicitadas, esta es eliminada definitivamente del sistema.

El sistema además comprueba si hay miembros con estado pendiente de rechazo que no tengan devoluciones pendientes. Estos miembros son rechazados definitivamente del proyecto.



Operación: completarDevolucionesPendientes(grupo: Grupo)
Descripción: Completa las devoluciones pendientes
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

El usuario participa como administrador en el grupo
grupo.creador.usuarioRegistrado = self

Postcondiciones:
Comprueba si hay devoluciones pendientes completada. Las que no se han completado, vuelve a transmitir la información para completarlas. Con las devoluciones completadas realiza los cambios necesarios en el sistema
let devoluciones: TupleType(completadas: Set(Devolución), sinCompletar: Set(Devolución)) =
estanCompletadasDevolucionesPendientes(grupo.devolución) in

Intenta completar las devoluciones aun pendientes
let devolucionesCompletadas : Set(Devolución) = accionarDevoluciones(devoluciones.sinCompletar) in
devolucionesCompletadas
-> union(devoluciones.completadas)
-> iterate(d ; infoUsuario : InfoUsuario = d.usuarioRegistrado.userInfo -> select(u | u.grupo = grupo) |
infoUsuario.devolucionPendiente = userInfo.devolucionPendiente - d.cantidad and
grupo.recaudadoTotal = grupo.recaudadoTotal - d.cantidad and
if not d.recompensaSolicitada -> oclIsUndefined() and
d.recompensaSolicitada.estado = EstadoRecompensa::PendienteDeDevoluciónDePago then
d.recompensaSolicitada.recompensa.solicitadas = d.recompensaSolicitada.recompensa.solicitadas - 1 and
d.recompensaSolicitada.recompensa.recompensaSolicitada -> excludes(d.recompensaSolicitada) and
RecompensaSolicitada.allInstances() -> excludes(d.recompensaSolicitada)
if not d.recompensaSolicitada.recompensa.activada and d.recompensaSolicitada.recompensa.solicitadas - 1 then
grupo.recompensa -> excludes(d.recompensaSolicitada.recompensa) and
Recompensa.allInstances() -> excludes(d.recompensaSolicitada.recompensa)
endif
endif
Devoluciones.allInstances() -> excludes(d)
)
grupo.miembroAceptado
-> select(m | m.estado = EstadoUsuario::PendienteRechazo and m.usuarioRegistrado.devolución -> isEmpty())
-> forAll(m | grupo.miembroRechazado -> includes(m) and grupo.miembroAceptado -> excludes(m) and
m.estado = EstadoUsuario::Rechazado and grupo.evento.miembroParticipa -> excludes(m) and
m.fecha = fechaActual() and grupo.conexiónChat.miembro -> excludes(m) and
grupo.totalMiembros = grupo.totalMiembros - 1)

Operación: estanCompletadasDevolucionesPendientes(devoluciones: Set(Devolución)):
TupleType(completadas: Set(Devolución), sinCompletar: Set(Devolución))
Descripción: Comprueba si están completadas las devoluciones
Precondiciones:
Deben existir las devoluciones en el sistema
Devolución.allInstances() -> includesAll(devoluciones)

El usuario participa como administrador en el grupo de cada devolución
devoluciones -> forAll(d | d.grupo.creador.usuarioRegistrado = self)

Salida:
Retorna las devoluciones completadas y sin completar
let devoluciones = Tuple{completadas = devolucionesCompletadas, sinCompletar = devolucionesSinCompletar)

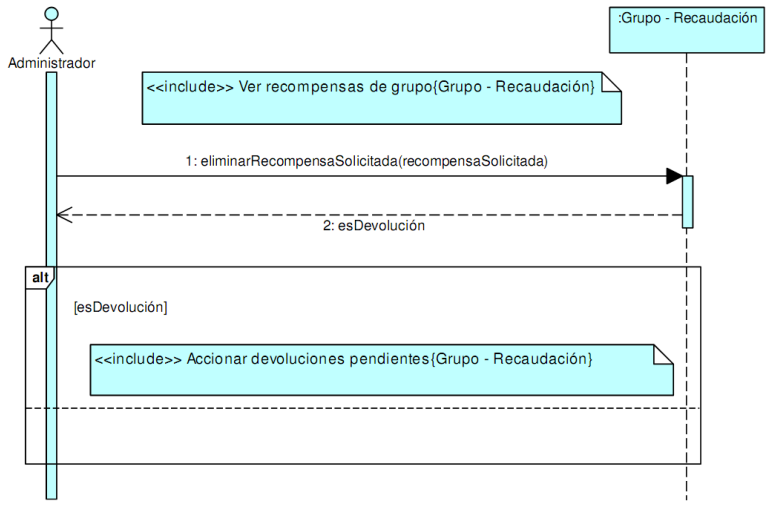
Operación: accionarDevoluciones(devolucionesSinCompletar: Set(Devolución)): Boolean
Descripción: Comprueba si están completadas las devoluciones
Precondiciones:
Deben existir las devoluciones en el sistema
Devolución.allInstances() -> includesAll(devolucionesSinCompletar)

El usuario participa como administrador en el grupo de cada devolución
devolucionesSinCompletar -> forAll(d | d.grupo.creador.usuarioRegistrado = self)

Salida:
Comunica a la empresa gestora de transacciones las devoluciones a realizar y devuelve las devoluciones completadas.

Eliminar recompensa solicitada vista grupo

El administrador elimina una recompensa con pago pendiente o completado. Si la recompensa tiene el pago pendiente este también queda eliminado. En caso de que fuera un pago pendiente de un miembro pendiente de aceptación, este pasa a ser aceptado. Sin embargo, si la recompensa tiene el pago completado, este pago debe haberse realizado hace menos de 60 días según establece el gestor de transacciones para poder realizar la devolución del pago. En cualquier caso una vez eliminada la recompensa solicitada, es comprobado si la recompensa estaba activada. Si no estuviera activada y ya no tuviera recompensas solicitadas asociadas es eliminada del sistema definitivamente.



Operación: eliminarRecompensaSolicitada(recompensaSolicitada: RecompensaSolicitada): Boolean
Descripción: Elimina una recompensa solicitada del proyecto.
Precondiciones:

Debe existir la recompensa solicitada en el sistema
RecompensaSolicitada.allInstances() -> exists(r | r = recompensaSolicitada)

El usuario participa como administrador en el proyecto al que pertenece la recompensa solicitada
recompensaSolicitada.recompensa.grupo.creador.usuarioRegistrado = self

La recompensa solicitada tiene el pago completado o pendiente
recompensaSolicitada.transacción.oclIsTypeOf(PagoPendiente) or
recompensaSolicitada.transacción.oclIsTypeOf(PagoCompletado)

La recompensa solicitada tiene el estado Pendiente de pago, Pago completado o Pendiente de envío
recompensaSolicitada.estado = EstadoRecompensa::PendienteDePago or
recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado or
recompensaSolicitada.estado = EstadoRecompensa::PendienteDeEnvío

Si la recompensa tiene el pago completado este se ha realizado hace menos de 60 días
if recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado then
fechaActual() - recompensaSolicitada.PagoCompletado.fecha <= 60
endif

Postcondiciones:

Elimina la recompensa solicitada del proyecto siempre y cuando tenga el pago pendiente, sino pasará a estar pendiente de devolución.En caso de ser un pago pendiente y el usuario miembro pendiente de aceptación del proyecto, este pasará a ser aceptado.

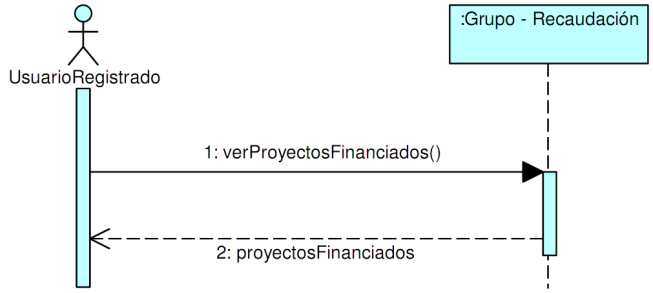
```
let infoUsuario: InfoUsuario =
        recompensaSolicitada.recompensa.grupo.infoUsuario -> select(u | u.usuarioRegistrado = self) in
if recompensaSolicitada.transacción.oclIsTypeOf(PagoPendiente) then
  infoUsuario.pagoPendiente = 0
  if infoUsuario.membresía = TipoUsuario::Miembro and infoUsuario.estado = EstadoUsuario::PendienteAceptación then
    infoUsuario.estado = EstadoUsuario::Aceptado
  endif
  Transacción.allInstances() -> excludes(recompensaSolicitada.transacción)
  RecompensaSolicitada.allInstances() -> excludes(recompensaSolicitada)
else
  infoUsuario.inversión = infoUsuario.inversión - recompensaSolicitada.pagoCompletado.cantidad
  infoUsuario.devoluciónPendiente =
        infoUsuario.devoluciónPendiente + recompensaSolicitada.pagoCompletado.cantidad and
  recompensaSolicitada.transacción.oclIsType(Devolución) and
  recompensaSolicitada.transacción.estado = 'Sin realizar' and
  recompensaSolicitada.estado = EstadoRecompensa::PendienteDeDevoluciónDePago
endif

if not recompensaSolicitada.recompensa.activada and d.recompensaSolicitada.recompensa.solicitadas - 1 then
  grupo.recompensa -> excludes(d.recompensaSolicitada.recompensa) and
  Recompensa.allInstances() -> excludes(d.recompensaSolicitada.recompensa)
endif
```

Salida:

```
let esDevolución : Boolean in

if recompensaSolicitada.transacción.oclIsTypeOf(PagoPendiente) then
  esDevolución = false
else
  esDevolución = true
endif
```

Ver proyectos financiados

Muestra los proyectos financiados por el usuario, por tanto, muestra los proyectos donde han habido pagos completados por parte del usuario. Junto al nombre del proyecto son mostrados también la cantidad, fecha, membresía dentro del proyecto y recompensa, si es que la solicitó.

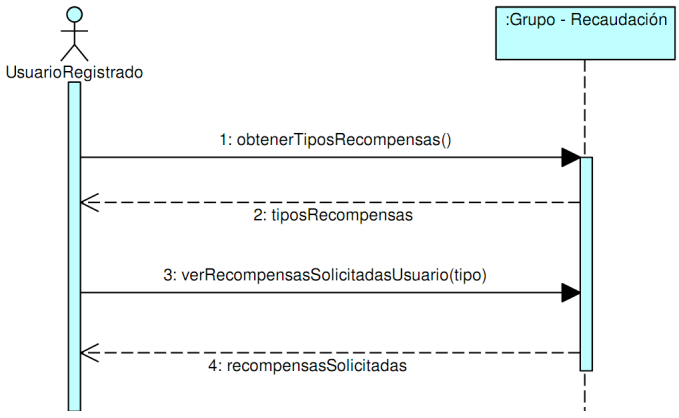


Operación: verProyectosFinanciados():
Set(TupleType(proyecto: Proyecto, cantidad: Integer, fecha: Fecha, membresía: TipoUsuario, recompensa: String))
Descipción: Genera el listado de proyectos financiados por el usuario
Precondiciones: -
Salida:
Retorna una colección de información sobre los proyectos financiados.

```
let proyectosFinanciados:
  Set( TupleType(proyecto: Proyecto, cantidad: Integer, fecha: Fecha, membresía: TipoUsuario, recompensa: String)) in

self.pagoCompletado -> forAll(p | proyectosFinanciados -> union(Tuple{proyecto = p.grupo.proyecto,
                                  cantidad = p.cantidad, fecha = p.fecha,
                                  membresía = self.infoUsuario -> intersection(p.grupo.infoUsuario).membresía,
                                  recompensa = p.recompensaSolicitada.recompensa.nombre}))
  proyectosFinanciados = proyectosFinanciados -> asSet()
```

Marcar recompensa solicitada como recibida

El usuario indica que ha recibido ya la recompensa que solicitó la cual tiene el estado pendiente de envío, y por tanto, el sistema puede eliminarla.



Operación: MarcarRecompensaSolicitadaComoRecibida(recompensaSolicitada: RecompensaSolicitada)
Descripción: Marca una recompensa solicitada por el usuario como recibida.
Precondiciones:

Debe existir la recompensa solicitada en el sistema
RecompensaSolicitada.allInstances() -> exists(r | r = recompensaSolicitada)

La recompensa solicitada pertence al usuario
recompensaSolicitada.transacción.usuarioRegistrado = self

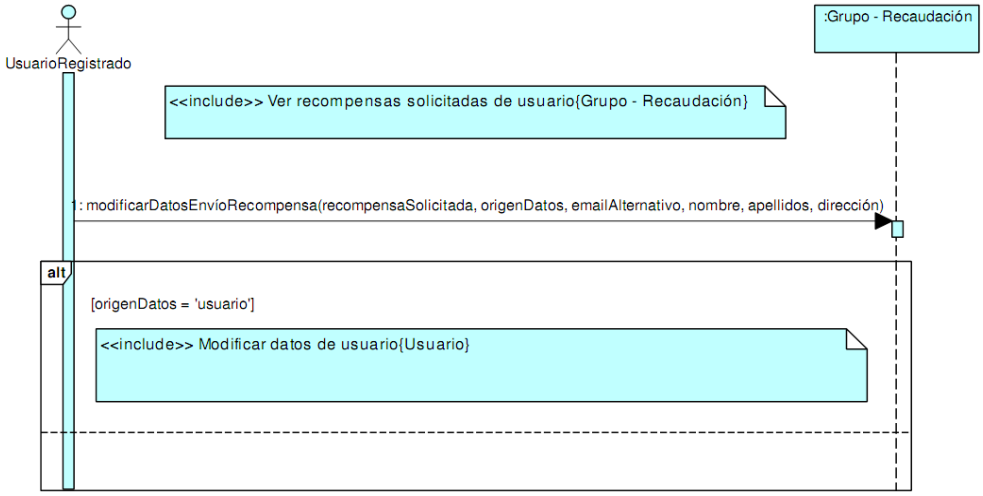
El pago de la recompensa solicitada es un pago completado
recompensaSolicitada.transacción.oclIsTypeOf(PagoCompletado)

La recompensa tiene el estado pendiente de enviar
recompensaSolicitada.estado = EstadoRecompensa::PendienteDeEnvío

Postcondiciones:
La recompensa solicitada es eliminada del sistema.

recompensaSolicitada.recompensa.solicitadas = recompensaSolicitada.recompensa.solicitadas - 1
if not recompensaSolicitada.recompensa.activada and recompensaSolicitada.recompensa.solicitadas = 0 then
recompensaSolicitada.recompensa.grupo -> oclIsUndefined and
Recompensa.allInstances() -> excludes(recompensaSolicitada.recompensa)
endif
recompensaSolicitada.recompensa -> oclIsUndefined and
RecompensaSolicitada.allInstances() -> excludes(recompensaSolicitada)

Modifica los datos y/o el origen de estos de una recompensa solicitada por el usuario.



Operación: modificarDatosEnvíoRecompensa(recompensaSolicitada: RecompensaSolicitada,
origenDatos : OrigenDatosEnvíoRecompensa, emailAlternativo : String,
nombre : String, apellidos : String, dirección: String)
Descripción: Modifica los datos de envio de una recompensa solicitada.
Precondiciones:
Debe existir la recompensa solicitada en el sistema
RecompensaSolicitada.allInstances() -> exists(r | r = recompensaSolicitada)

La recompensa solicitada pertence al usuario
recompensaSolicitada.transacción.usuarioRegistrado = self

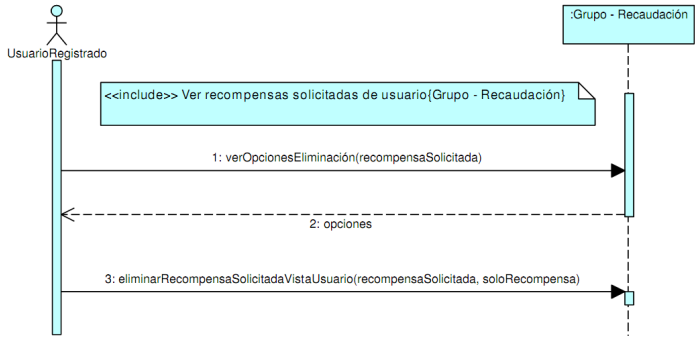
La recompensa no tiene el estado como pendiente de devolución
recompensa.estado <> EstadoRecompensa::PendienteDeDevoluciónDePago

Postcondiciones:
Modifica los datos de envio de la recompensa que el usuario haya indicado

```
if origenDatos -> notEmpty() then
  recompensaSolicitada.origenDatosEnvio = origenDatos
endif

if recompensaSolicitada.origenDatosEnvio = OrigenDatosEnvíoRecompensa::RecompensaSolicitada then
  let datosEnvio : TupleType(email : String, nombre : String, apellidos : String, dirección : String) in
  if not emailAlternativo -> oclIsUndefined() then
    datosEnvio.email = emailAlternativo
  else
    datosEnvio.email = recompensaSolicitada.datosEnvio.email
  endif
  if not nombre -> oclIsUndefined() then
    datosEnvio.nombre = nombre
  else
    datosEnvio.nombre = recompensaSolicitada.datosEnvio.nombre
  endif
  if not apellidos -> oclIsUndefined() then
    datosEnvio.apellidos = apellidos
  else
    datosEnvio.apellidos = recompensaSolicitada.datosEnvio.apellidos
  endif
  if not dirección -> oclIsUndefined() then
    datosEnvio.dirección = dirección
  else
    datosEnvio.dirección = recompensaSolicitada.datosEnvio.dirección
  endif
  recompensaSolicitada.datosEnvio = datosEnvio
endif
```

Eliminar recompensa solicitada vista usuario

Elimina una recompensa solicitada por parte del usuario. En caso de que la recompensa tenga el pago pendiente podrá elegir entre eliminar solo la recompensa o también el pago pendiente. En caso de ser un pago completado solo tendrá la opción de eliminar la recompensa pero el pago completado no se devolverá. Esto es así ya que quien tiene que accionar la devolución es el proyecto no el usuario y por tanto, si quisiera anular el pago tendrá que acordarlo con el administrador del proyecto para que él elimine la recompensa y accione la devolución.



Operación: verOpcionesEliminación(recompensaSolicitada: RecompensaSolicitada):
TupleType(soloRecompensa: Boolean, recompensaAdemasDeAportación: Boolean)
Descripción: Muestra las opciones de eliminación de la recompensa solicitada por el usuairo.
Precondiciones:

Debe existir la recompensa solicitada en el sistema
RecompensaSolicitada.allInstances() -> exists(r | r = recompensaSolicitada)

La recompensa solicitada pertence al usuario
recompensaSolicitada.transacción.usuarioRegistrado = self

El pago de la recompensa solicitada es un pago pendiente o completado
recompensaSolicitada.transacción.oclIsTypeOf(PagoPendiente) or
recompensaSolicitada.transacción.oclIsTypeOf(PagoCompletado)

La recompensa está en estado de pago pendiente o completado pero no enviado ni en devolución
recompensaSolicitada.estado = EstadoRecompensa::PendienteDePago or
recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado

Salida:

Muestra las opciones disponibles para eliminar la recompensa

```
let opciones : TupleType(soloRecompensa: Boolean, recompensaAdemasDeAportación: Boolean) in
let infoUsuario : InfoUsuario = recompensaSolicitada.recompensa.grupo.infoUsuario
                                                    -> select(u | u.usuarioRegistrado = self) in
if recompensa.estado = EstadoRecompensa::PendienteDePago then
  if (infoUsuario.membresía = TipoUsuario::Invitado and infoUsuario.estado = EstadoUsuario::Aceptado) or
    (infoUsuario.membresía = TipoUsuario::Miembro and infoUsuario.estado = EstadoUsuario::PendienteAceptación) then
      opciones = Tuple{soloRecompensa = true, recompensaAdemasDeAportación = false}
  else
      opciones = Tuple{soloRecompensa = true, recompensaAdemasDeAportación = true}
  endif
elseif recompensa.estado = EstadoRecompensa::PagoCompletado then
   opciones = Tuple{soloRecompensa = true, recompensaAdemasDeAportación = false}
endif
```

Operación: eliminarRecompensaSolicitadaVistaUsuario(recompensaSolicitada: RecompensaSolicitada,
soloRecompensa: Boolean)
Descripción: Elimina una recompensa solicitada por el mismo usuario
Precondiciones:
Debe existir la recompensa solicitada en el sistema
RecompensaSolicitada.allInstances() -> exists(r | r = recompensaSolicitada)

La recompensa solicitada pertence al usuario
recompensaSolicitada.transacción.usuarioRegistrado = self

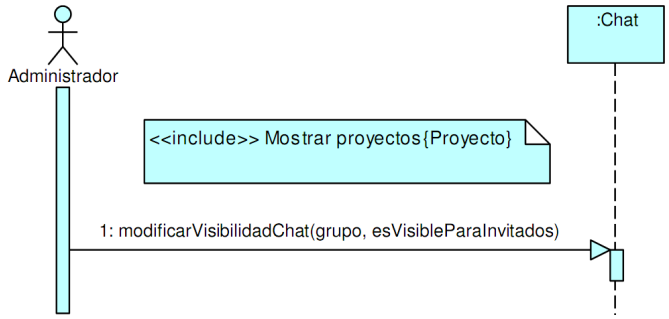
El pago de la recompensa solicitada es un pago pendiente o completado
recompensaSolicitada.transacción.oclIsTypeOf(PagoPendiente) or
recompensaSolicitada.transacción.oclIsTypeOf(PagoCompletado)

La recompensa está en estado de pago pendiente o completado pero no enviado ni en devolución
recompensaSolicitada.estado = EstadoRecompensa::PendienteDePago or
recompensaSolicitada.estado = EstadoRecompensa::PagoCompletado
Postcondiciones:
La recompensa es eliminada junto con la aportación si así se indica y está permitido
let infoUsuario : InfoUsuario = recompensaSolicitada.recompensa.grupo.infoUsuario
-> select(u | u.usuarioRegistrado = self) in
if recompensaSolicitada.estado = EstadoRecompensa::PendienteDePago and
not(infoUsuario.membresía = TipoUsuario::Invitado and infoUsuario.estado = EstadoUsuario::Aceptado) and
not (infoUsuario.membresía = TipoUsuario::Miembro and infoUsuario.estado = EstadoUsuario::PendienteAceptación) and
not soloRecompensa then
PagoPendiente.allInstances() -> excludes(recompensaSolicitada.pagoPendiente)
infoUsuario.pagoPendiente = 0
endif
recompensaSolicitada.recompensa.solicitadas = recompensaSolicitada.recompensa.solicitadas - 1
if not recompensaSolicitada.recompensa.activada and recompensaSolicitada.recompensa.solicitadas = 0 then
recompensaSolicitada.recompensa.grupo -> oclIsUndefined and
Recompensa.allInstances() -> excludes(recompensaSolicitada.recompensa)
endif
recompensaSolicitada.recompensa -> oclIsUndefined and
RecompensaSolicitada.allInstances() -> excludes(recompensaSolicitada)

6. Diagramas de secuencia chat

Modificar visibilidad chat

Modifica la visibilidad de los mensajes de chat de forma que pueda elegirse cuando tener una conversación a través de este medio, cerrada solo para administrador y miembros o abierta también para invitados.



Operación: modificarVisibilidadChat(grupo: Grupo, esVisibleParaInvitados: Boolean)
Descripción: El administrador modifica la visibilidad de los mensajes del chat.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

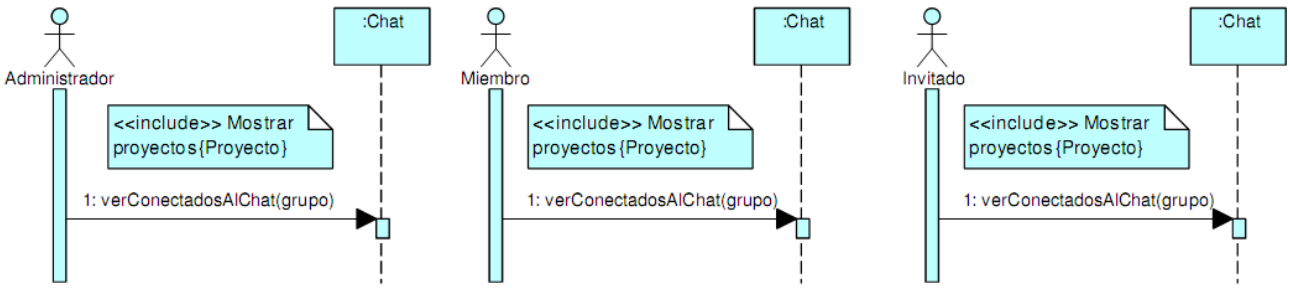
El administrador lo es del propio grupo
grupo.creador.usuarioRegistrado = self

Postcondiciones:
Modifica la visibilidad para que sea visible solo a administrador y miembros o a administrador, miembros y también invitados.

```
if esVisibleParaInvitados then
    grupo.chat.visibilidad = VisibilidadChat::VisibleParaAdministradorMiembrosInvitados
else
    grupo.chat.visibilidad = VisibilidadChat::VisibleParaAdministradorMiembros
endif
```

Ver conectados

Muestra el listado de conectados al chat junto con su membresía.



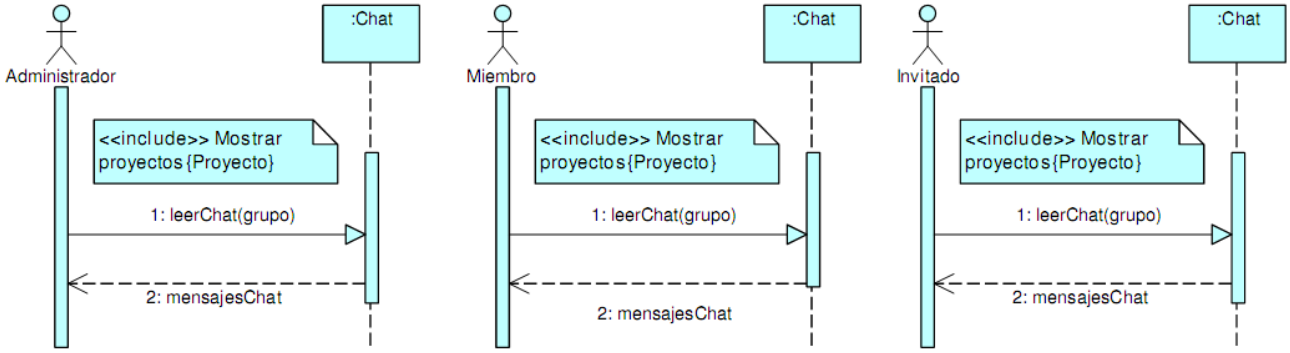
Operación: verConectadosAlChat(grupo: Grupo): Set(TupleType(usuario: UsuarioRegistrado, membresía: TipoUsuario))
Descripción: Muestra los conectados al chat en un instante de tiempo
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Participa como administrador, miembro o invitado del proyecto
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self) or
grupo.invitadoAceptado.usuarioRegistrado -> includes(self)
Salida:
Genera el listado de conectados al chat en este instante

```
let usuariosConectados : Set(TupleType(usuario: UsuarioRegistrado, membresía: TipoUsuario)) in
if grupo.creador.conexiónChat.chat -> includes(grupo.chat) then
  usuariosConectados -> includes(Tuple{usuario = grupo.creador.usuarioRegistrado,
                                       membresía = TipoUsuario::Administrador})
endif
grupo.miembroAceptado -> select(m | m.conexiónChat.chat -> includes(grupo.chat))
      -> iterate(m | usuariosConectados -> includes( Tuple{usuario = m.usuarioRegistrado,
                                                            membresía = TipoUsuario::Miembro}))
grupo.invitadoAceptado -> select(i | i.conexiónChat.chat -> includes(grupo.chat))
      -> iterate(i | usuariosConectados -> includes( Tuple{usuario = i.usuarioRegistrado,
                                                            membresía = TipoUsuario::Invitado}))
```

Leer chat

Muestra los mensajes del chat del proyecto ordenados de más antiguos a más recientes, junto con el autor del mensaje y la fecha.



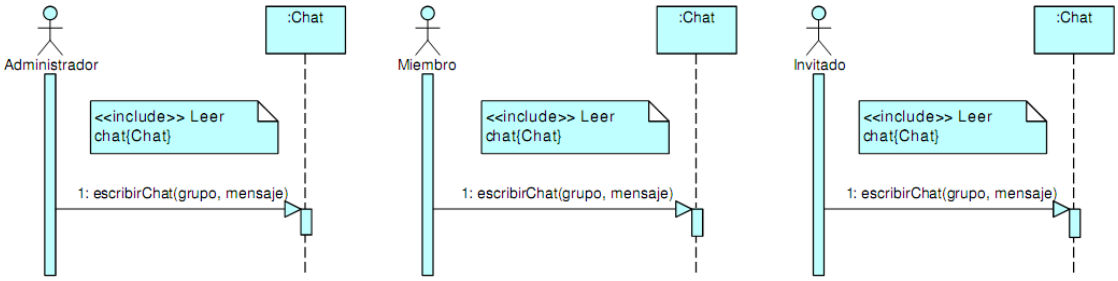
Operación: leerChat(grupo: Grupo): Set(TupleType(mensaje: mensajeChat, autor: UsuarioRegistrado))
Descripción: Obtiene los mensajes del chat de un proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Participa como administrador, miembro o invitado del proyecto
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self) or
grupo.invitadoAceptado.usuarioRegistrado -> includes(self)
PostCondiciones:
El usuario se conecta al chat
let infoUsuario : InfoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
if infoUsuario.conexiónChat.chat -> excludes(grupo.chat) then
infoUsuario.conexiónChat.chat -> includes(grupo.chat)
endif
Salida:
Si particia como invitado solo obtendrá los mensajes creados con visibilidad pública. Tanto administrador como miembros verán todos los mensajes. Los mensajes estarán ordenados de más antiguo a más reciente.

```
let mensajesChat : Set(TupleType(mensaje: mensajeChat, autor: UsuarioRegistrado)) in
let infoUsuario : InfoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
if infoUsuario.oclIsTypeOf(Invitado) then
  let mensajes : Set(MensajeChat) = grupo.chat.mensajeChat
                    -> select(m | m.visibilidad = VisibilidadChat::VisibleParaAdministradorMiembrosInvitados) in
  mensajes -> iterate(m | mensajesChat -> includes(Tuple{mensaje = m, autor = m.infoUsuario.usuarioRegistrado}))
else
  grupo.chat.mensajeChat
          -> iterate(m | mensajesChat -> includes(Tuple{mensaje = m, autor = m.infoUsuario.usuarioRegistrado}))
endif
mensajesChat = mensajesChat -> sortedBy(fecha)
```

Escribir chat

Escribe un nuevo mensaje de chat que será visible por el resto de usuarios conectados a éste. Este mensaje además tendrá la visibilidad que el mismo chat tiene en ese instante.



Operación: escribirChat(grupo: Grupo, mensaje: String)
Descripción: Escribe un mensaje de chat de un proyecto
Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

Participa como Invitado, Miembro o Administrador del proyecto

```
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self) or
grupo.invitadoAceptado.usuarioRegistrado -> includes(self)
```

Si participa como invitado la visibilidad actual del chat debe ser tanto para administrador, miembros como invitados

```
let infoUsuario : InfoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
if grupo.chat.visibilidad = VisibilidadChat::VisibleParaAdministradorMiembros then
   infoUsuario.oclIsTypeOf(Administrador) or infoUsuario.oclIsTypeOf(Miembro)
endif
```

El mensaje no está vacío tiene un tamaño mínimo de 1 caracter y máximo de 100 caracteres

```
not mensaje -> oclIsUndefined() and mensaje -> size() >= 1 and mensaje -> size() <= 100
```

PostCondiciones:

El usuario se conecta al chat

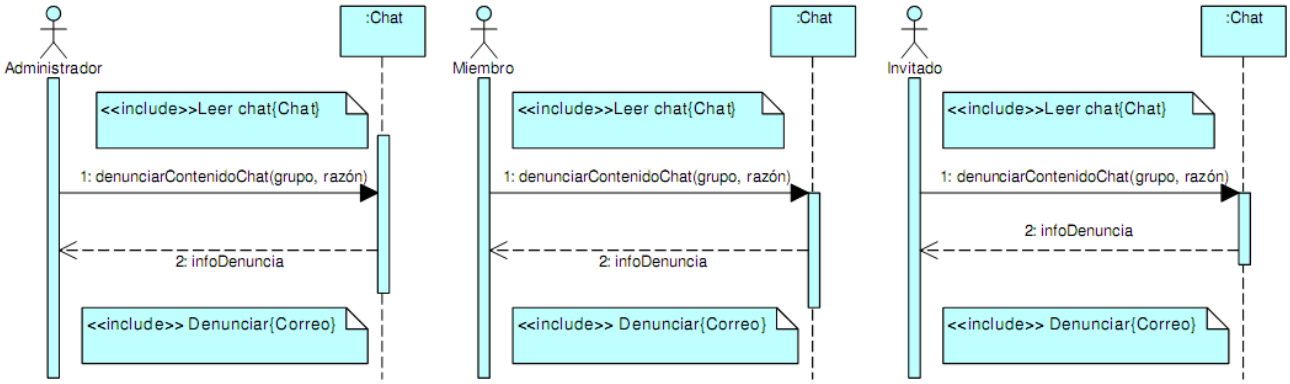
```
let infoUsuario : InfoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
if infoUsuario.conexiónChat.chat -> excludes(grupo.chat) then
   infoUsuario.conexiónChat.chat -> includes(grupo.chat)
endif
```

Un mensaje de chat es creado asociado con el chat del proyecto y el usuario que lo ha escrito.
La visibilidad del mensaje será la misma que la del chat en el instante en que el mensaje es creado.

```
let infoUsuario : InfoUsuario = self.infoUsuario -> select(i | i.grupo = grupo) in
let mensajeChat : MensajeChat -> oclIsUndefined() in
mensajeChat.oclIsNew() and mensajeChat.oclIsTypeOf(MensajeChat) and mensajeChat.fecha = fechaActual() and
mensajeChat.cuerpo = mensaje and mensajeChat.chat = grupo.chat and mensajeChat.infoUsuario = infoUsuario and
mensajeChat.visibilidad = grupo.chat.visibilidad and
mensajeChat.identificador = MensajeChat.allInstances().identificador -> max() + 1
```

Denunciar chat

El usuario denuncia el contenido del chat indicando una razón.



Operación: denunciarContenidoChat(grupo: Grupo, razón: String): TupleType(razón: String, infoExtra: String)
Descripción: Obtiene los datos necesarios para formalizar la denuncia
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Participa como Invitado, Miembro o Administrador del proyecto
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self) or
grupo.invitadoAceptado.usuarioRegistrado -> includes(self)

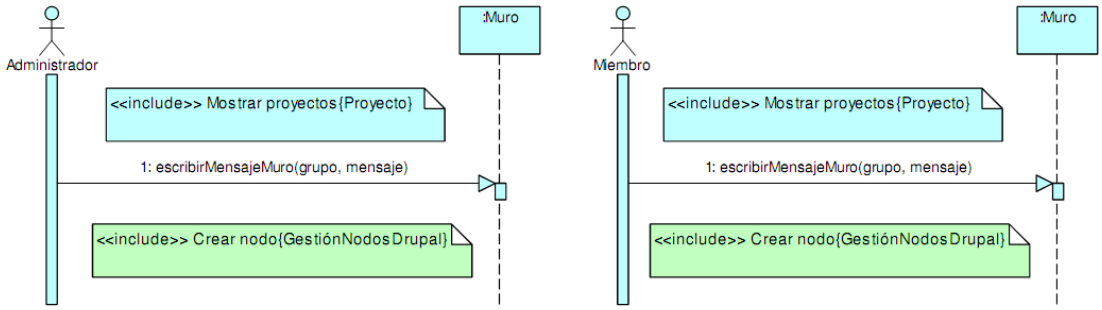
Salida:
Se recopila la información necesaria para enviar.

```
let mensajesChat : String -> oclIsUndefined in
grupo.chat.mensajeChat
    -> iterate(m | mensajesChat  =  menajesChat -> concat(m.infoUsuario.usuarioRegistrado)
                                                    -> concat(' : ') -> concat(m.cuerpo)))
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{ razón = razón,
           InfoExtra = 'Tipo contenido: Chat, Identificador: '  -> concat(grupo.identificador) -> concat(', Contenido: ')
                    -> concat(mensajesChat)}
```

6. Diagramas de secuencia muro

Escribir muro

El administrador o miembro escribe un mensaje en el muro.



Operación: escribirMensajeMuro(grupo: Grupo, mensaje: String)
Descripción: El administrador o miembro escribe un mensaje en el muro del proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

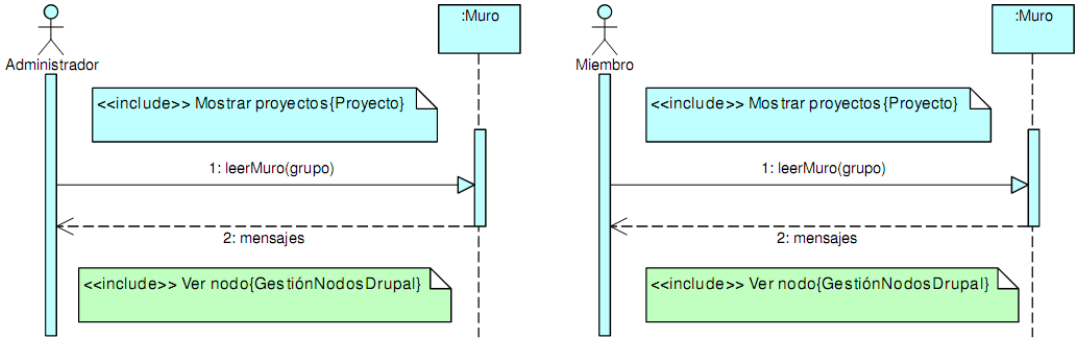
Participa como Administrador o Miembro del proyecto
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado = self

El mensaje no está vacío y tiene un tamaño mínimo de 1 caracter y máximo de 2000 caracteres
not mensaje -> oclIsUndefined() and mensaje -> size() >= 1 and mensaje -> size() <= 2000

Postcondiciones:
El mensaje queda registrado en el muro.
let infoUsuario : InfoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
let mensajeMuro : MensajeMuro -> oclIsUndefined() in
mensajeMuro.oclIsNew() and mensajeMuro.oclIsTypeOf(MensajeMuro) and mensajeMuro.fecha = fechaActual() and
mensajeMuro.cuerpo = mensaje and mensajeMuro.muroMensajes = grupo.muroMensajes and
mensajeMuro.infoUsuario = infoUsuario

Leer muro

El administrador o miembro solicitan los mensajes publicados en el muro de un proyecto. Cada mensaje de muro contiene además del propio mensaje, la fecha de publicación, el autor del mismo y el listado de comentarios asociados a este. Los mensajes son ordenados de más recientes a más antiguos y los comentarios de más antiguos a más recientes.



Operación: leerMuro(grupo: Grupo): Set(TupleType(autorMensaje: UsuarioRegistrado, mensaje: MensajeMuro, comentarios: Set(TupleType(autorComentario: UsuarioRegistrado, comentario: ComentarioMuro))))

Descripción: Muestra los mensajes de muro de un proyecto junto con los comentarios de éstos.

Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

Participa como Administrador o Miembro del proyecto

```
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado = self
```

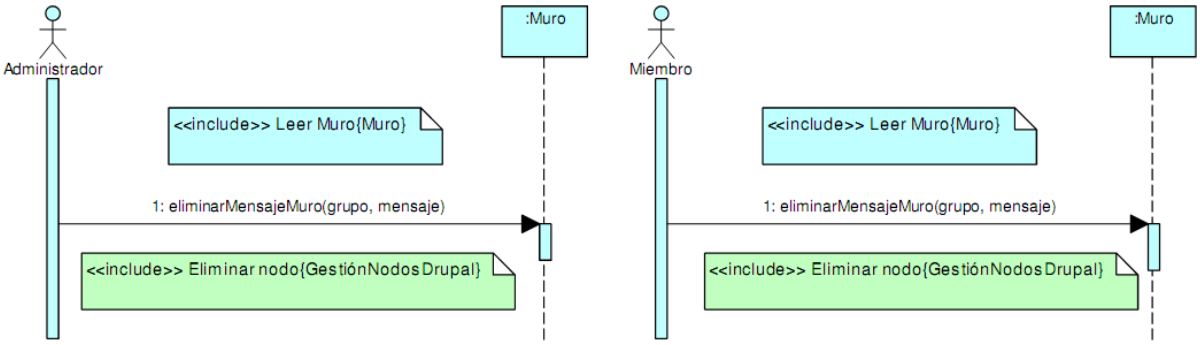
Salida:

Genera el listado de mensajes de muro ordenados de más reciente a más antiguos. Cada mensaje tiene asociados loscomentarios que le pertenecen ordenados de más antiguo a más reciente

```
let mensajes : Set(TupleType(autorMensaje: UsuarioRegistrado, mensaje: MensajeMuro,
                    comentarios: Set(TupleType(autorComentario: UsuarioRegistrado, comentario: ComentarioMuro)))) in
 grupo.muroMensajes.mensajeMuro
  -> iterate(m ; comentarios : Set(TupleType(autorComentario: UsuarioRegistrado, comentario: ComentarioMuro)) -> isEmpty() |
       m.comentarioMuro -> iterate(c | comentarios -> includes(Tuple{autorComentario = c.infoUsuario.usuarioRegistrado,
                                                                  comentario = c})
                                  )
     and comentarios = comentarios->sortedBy(c | c.comentario.fecha) and
     mensajes -> includes(Tuple{autorMensaje = m.infoUsuario.usuarioRegistrado, mensaje = m, comentarios = comentarios})
  )
  and mensajes -> sortedBy(m | m.mensaje.fecha) -> reverse()
```

Eliminar mensaje muro

El administrador o miembro elimina un mensaje del muro. En caso de ser un miembro este debe ser el autor del propio mensaje, en cambio el administrador puede borrar cualquiera.



Operación: eliminarMensajeMuro(grupo: Grupo, mensaje: mensajeMuro)
Descripción: Elimina un mensaje del muro junto cons sus comentarios asociados.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el mensaje en el sistema y pertenecer al mismo grupo que su muro de mensajes
MensajeMuro.allInstances() -> exists(m | m = mensaje) and mensaje.muroMensajes = grupo.muroMensajes

Participa como Administrador o Miembro del proyecto
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self)

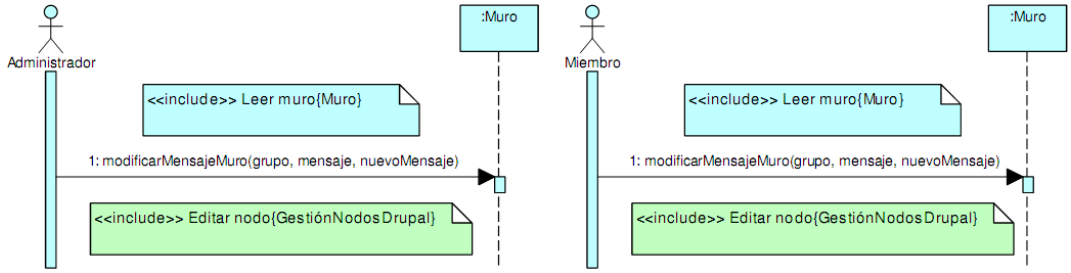
Si es Miembro deberá ser el autor del mensaje
if grupo.miembroAceptado.usuarioRegistrado -> includes(self) then
mensaje.infoUsuario.usuarioRegistrado = self
endif

Postcondiciones:
El mensaje queda eliminado del sistema junto con sus comentarios.

let comentarios : Set(Comentario) = mensaje.comentarioMensaje in
comentarios -> forAll(c | c.mensajeMuro -> oclIsUndefined() and c.infoUsuario -> oclIsUndefined()) and
ComentarioMensaje.allInstances() -> excludesAll(comentarios) and mensaje.infoUsuario -> oclIsUndefined() and
mensaje.muroMensajes -> oclIsUndefined() and MensajeMuro.allInstances() -> excludes(mensaje)

Editar mensaje muro

El usuario edita un mensaje de muro del cual es autor.



Operación: modificarMensajeMuro(grupo: Grupo, mensaje: mensajeMuro, nuevoMensaje: String)
Descripción: Edita un mensaje del muro.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el mensaje en el sistema y pertenecer al mismo grupo que su muro de mensajes
MensajeMuro.allInstances() -> exists(m | m = mensaje) and mensaje.muroMensajes = grupo.muroMensajes

Participa como Administrador o Miembro del proyecto
grupo.creador.usuarioRegistrado = self or
grupo.miembroAceptado.usuarioRegistrado -> includes(self)

El usuario deberá ser el autor del mensaje
mensaje.infoUsuario.usuarioRegistrado = self

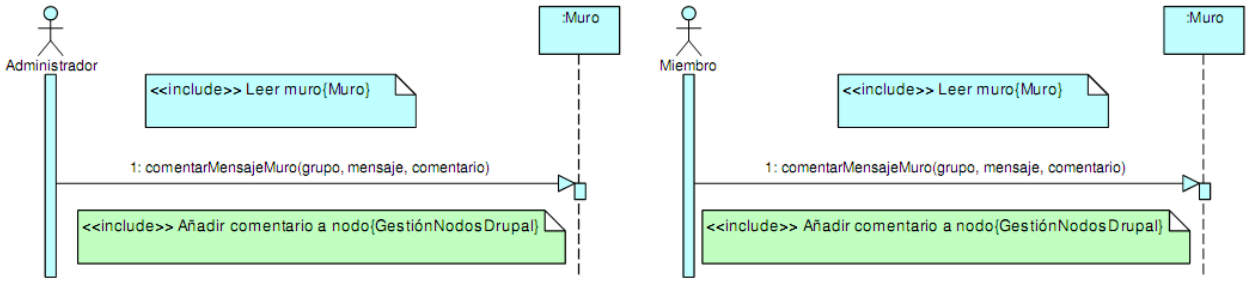
El nuevoMensaje no está vacío y tiene un tamaño mínimo de 1 caracter y máximo de 2000 caracteres
not nuevoMensaje -> oclIsUndefined() and nuevoMensaje -> size() >= 1 and nuevoMensaje -> size() <= 2000

Postcondiciones:
El mensaje es modificado y son guardados los cambios en el sistema.

mensaje.cuerpo = nuevoMensaje

Comentar mensaje

El administrador o miembro comenta un mensaje del muro del proyecto .



Operación: comentarMensajeMuro(grupo: Grupo, mensaje: mensajeMuro, comentario: String)
Descripción: Escribe un comentario a un mensaje del muro.
Precondiciones:

Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el mensaje en el sistema y pertenecer al mismo grupo que su muro de mensajes
MensajeMuro.allInstances() -> exists(m | m = mensaje) and mensaje.muroMensajes = grupo.muroMensajes

Participa como Administrador o Miembro del proyecto
grupo.creador.usuarioRegistrado = self orgrupo.miembroAceptado.usuarioRegistrado -> includes(self)

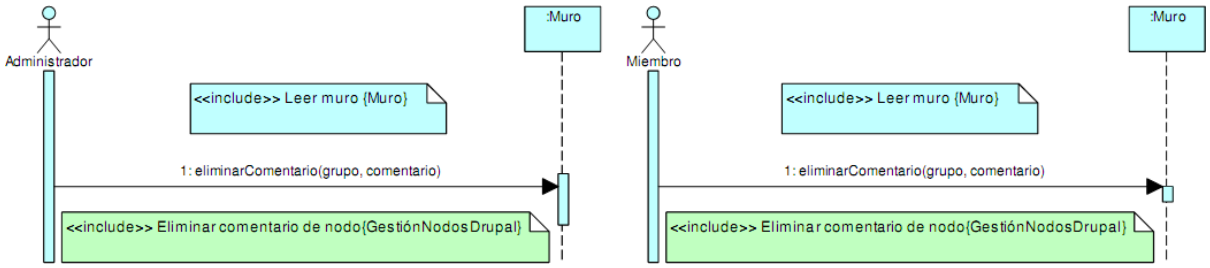
El comentario no está vacío y tiene un tamaño mínimo de 1 caracter y máximo de 500 caracteres
not comentario -> oclIsUndefined() and comentario -> size() >= 1 and comentario -> size() <= 500

Postcondiciones:
El comentario es asociado al mensaje de muro.

let infoUsuario : InfoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
let comentarioMuro : ComentarioMuro ->oclIsUndefined() in
comentarioMuro.oclIsNew() and comentarioMuro.oclIsTypeOf(ComentarioMuro) and
comentarioMuro.fecha = fechaActual() and comentarioMuro.cuerpo = comentario and
comentarioMuro.infoUsuario = infoUsuario and comentarioMuro.mensajeMuro = mensajeMuro

Eliminar comentario

El administrador o miembro elimina un comentario asociado a un mensaje del muro. En caso de participar como miembro tiene que ser el autor del comentario o del mensaje que lo contiene para poderlo eliminar.



Operación: eliminarComentarioMensaje(grupo: Grupo, comentario: ComentarioMuro)
Descripción: Elimina un comentario de un mensaje del muro.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el comentario en el sistema y pertenecer al mismo grupo que su muro de mensajes
ComentarioMuro.allInstances() -> exists(c | c = comentario) and
comentario.mensajeMuro.muroMensajes = grupo.muroMensajes

Participa como Administrador o Miembro del proyecto
grupo.creador.usuarioRegistrado = self orgrupo.miembroAceptado.usuarioRegistrado -> includes(self)

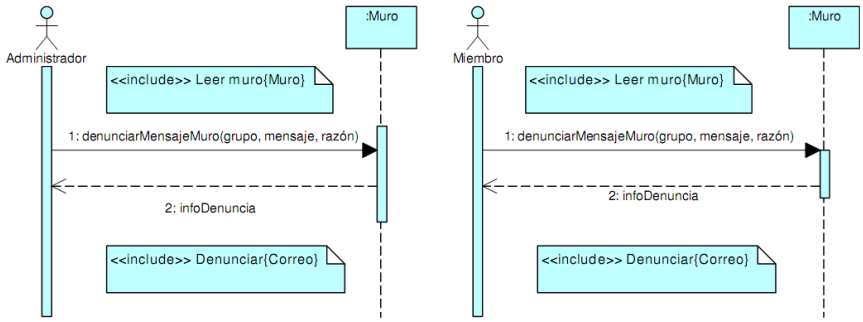
Si participa como Miembro, el usuario deberá ser el autor del comentario o del mensaje que lo contiene
if grupo.miembroAceptado.usuarioRegistrado -> includes(self) then
comentario.infoUsuario.usuarioRegistrado = self or
comentario.mensajeMuro.infoUsuario.usuarioRegistrado = self
endif

Postcondiciones:
El comentario queda eliminado del sistema.

comentario.infoUsuario -> oclIsUndefined() and comentario.mensajeMuro -> oclIsUndefined() and
ComentarioMuro.allInstances -> excludes(comentario)

Denunciar mensaje de muro

El usuario denuncia un mensaje de muro indicando una razón.



Operación: denunciarMensajeMuro(grupo: Grupo, mensaje: MensajeMuro, razón: String): TupleType(razón: String, infoExtra: String)
Descripción: Obtiene los datos necesarios para formalizar la denuncia
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Debe existir el mensaje de muro en el sistema y pertenecer al mismo grupo que su muro de mensajes
MensajeMuro.allInstances() -> exists(m | m = mensaje) and mensaje.muroMensajes = grupo.muroMensajes

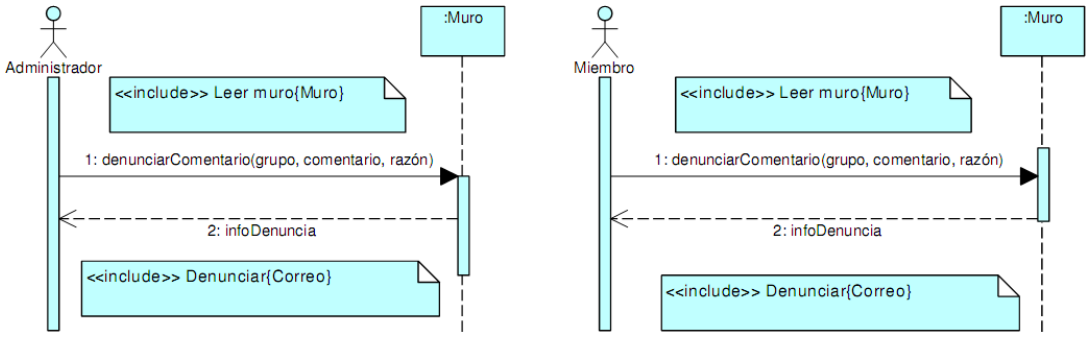
Participa como Administrador o Miembro del proyecto al que pertenece el mensaje de muro.
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self)

Salida:
Se recopila la información necesaria para enviar.

```
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{razón = razón,
        infoExtra = 'Tipo contenido: MensajeMuro, IdentificadorMensaje: '  -> concat(mensaje.identificador)
                -> concat(', IdentificadorGrupo: ') -> concat(evento.grupo.identificador)
```

Denunciar comentario

El usuario denuncia un comentario de un mensaje de muro indicando una razón.



Operación: denunciarComenario(grupo: Grupo, comentario: comentarioMuro, razón: String): TupleType(razón: String, infoExtra: String)

Descripción: Obtiene los datos necesarios para formalizar la denuncia

Precondiciones:

Debe existir el grupo en el sistema

```
Grupo.allInstances() -> exists(g | g = grupo)
```

Debe existir el comentario en el sistema y pertenecer al mismo grupo que su muro de mensajes

```
ComentarioMuro.allInstances() -> exists(c | c = comentario) and
comentario.mensajeMuro.muroMensajes = grupo.muroMensajes
```

Participa como Administrador o Miembro del proyecto al que pertenece el comentario

```
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self)
```

Salida:

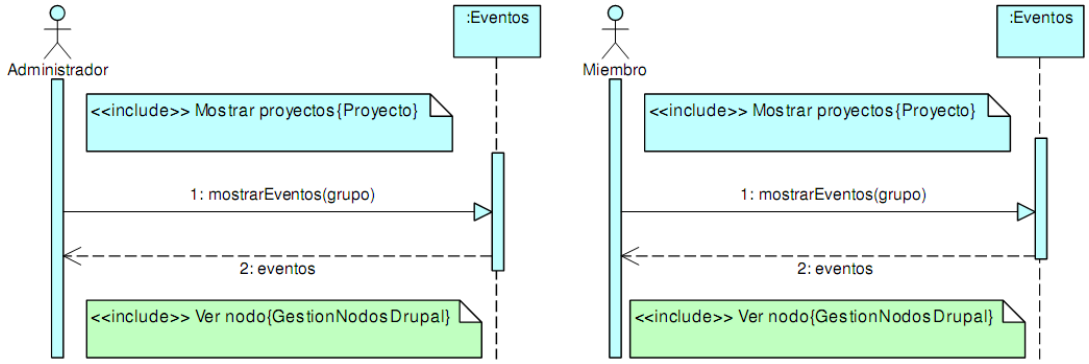
Se recopila la información necesaria para enviar.

```
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{razón: razón,
        infoExtra: 'Tipo contenido: Comentario, IdentificadorComentario: '  -> concat(comentario.identificador)
            -> concat(' , IdentificadorMensajeContenedor: '
            -> concat(comentario.mensajeMuro.identificador)
            -> concat(', IdentificadorGrupo: ') -> concat(evento.grupo.identificador)}
```

6. Diagramas de secuencia evento

Mostrar eventos

Muestra el listado de eventos del proyecto ordenados de más recientes a más antiguos.



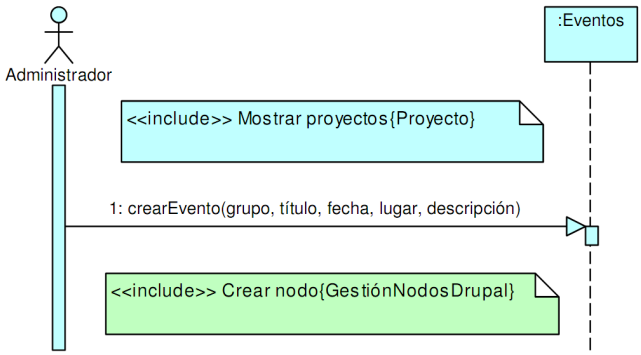
Operación: mostrarEventos(grupo: Grupo): Set(Evento)
Descripción: Muestra los eventos de un proyecto
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Participa como Administrador o miembro del proyecto
grupo.creador.usuarioRegistrado = self or grupo.miembroAceptado.usuarioRegistrado -> includes(self)

Salida:
Genera el listado de eventos del proyecto indicado
let eventos : Set(Evento) = grupo.evento in
evento = evento -> sortedBy(e | e.fecha) -> reverse()

Crear evento

El administrador del proyecto crea un evento compuesto por un título, fecha, lugar y descripción. En este evento podrán indicar su participación tanto el administrador como miembros del mismo proyecto.



Operación: crearEvento(grupo: Grupo, título: String, fecha: Fecha, lugar: String, descripción: String)
Descripción: Crea un evento para el grupo al que pertenece el administrador.
Precondiciones:
Debe existir el grupo en el sistema
Grupo.allInstances() -> exists(g | g = grupo)

Participa como Administrador del proyecto
grupo.creador.usuarioRegistrado = self

No hay datos vacíos
not título -> oclIsUndefined() and not fecha -> oclIsUndefined() and
not lugar -> oclIsUndefined() and not descripción -> oclIsUndefined()

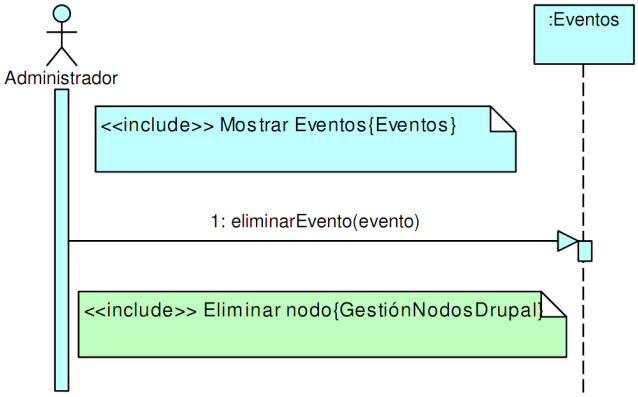
El título tiene entre 1 y 100 caracteres
título -> size() >= 1 and título -> size() <= 100

El lugar tiene entre 1 y 90 caracteres
lugar -> size() >= 1 and lugar -> size() <= 100

La descripción tiene entre 1 y 500 caracteres
descripción -> size() >= 1 and descripción -> size() <= 500

Postcondiciones:
Se crea un evento en el mismo grupo al que pertenece el administrador
let infoUsuario : InfoUsuario = grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
let evento : Evento -> oclIsUndefined in
evento.oclIsNew() and evento.oclIsTypeOf(Evento) and evento.título = título and evento.fecha = fecha and
evento.lugar = lugar and evento.descripción = descripción and grupo.evento -> includes(evento)

Eliminar evento



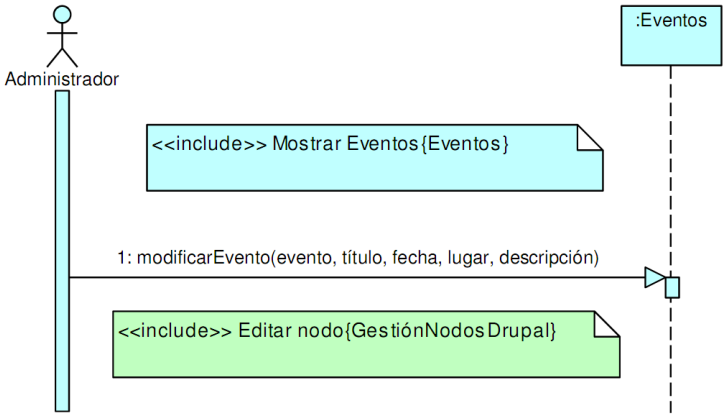
Operación: eliminarEvento(evento: Evento)
Descripción: El evento es eliminado.
Precondiciones:
Debe existir el evento en el sistema
Evento.allInstances() -> exists(e | e = evento)

Participa como Administrador del proyecto
evento.grupo.creador.usuarioRegistrado = self

Postcondiciones:
El evento es eliminado del sistema

evento.grupo.evento -> excludes(evento) and evento.administradorParticipa -> oclIsUndefined and evento.miembroParticipa -> isEmpty() and Evento.allInstances -> excludes(evento)

Modificar evento



Operación: modificarEvento(evento: Evento, título: String, fecha: Fecha, lugar: String, descripción : String)
Descripción: Modifica los datos del evento.
Precondiciones:

Debe existir el evento en el sistema

```
Evento.allInstances() -> exists(e | e = evento)
```

Participa como Administrador del proyecto

```
evento.grupo.creador.usuarioRegistrado = self
```

El título si no está vacío tiene entre 1 y 100 caracteres

```
título -> size() >= 1 and título -> size() <= 100
```

El lugar si no está vacío tiene entre 1 y 90 caracteres

```
lugar -> size() >= 1 and lugar -> size() <= 100
```

La descripción si no está vacía tiene entre 1 y 500 caracteres

```
descripción -> size() >= 1 and descripción -> size() <= 500
```

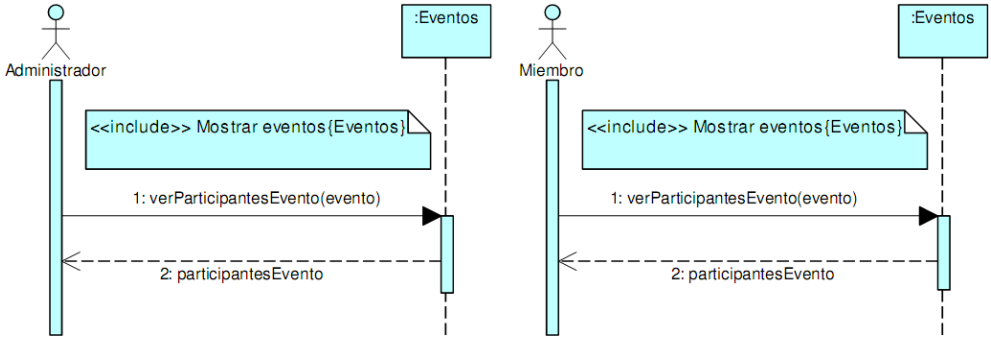
Postcondiciones:

El evento es modificado y los cambios son guardados en el sistema.

```
if not título -> oclIsUndefined() then
   evento.título = título
endif
if not fecha -> oclIsUndefined() then
   evento.fecha = fecha
endif
if not lugar-> oclIsUndefined() then
   evento.lugar = lugar
endif
if not descripción -> oclIsUndefined() then
   evento.descripción = descripción
endif
```

Ver participantes evento

Muestra los miembros y/o administrador que han indicado su deseo de participar en el evento publicado del proyecto.



Operación: verParticipantesEvento(evento: Evento): Set(UsuarioRegistrado)
Descripción: Muestra los participantes en un evento concreto
Precondiciones:
Debe existir el evento en el sistema
Evento.allInstances() -> exists(e | e = evento)

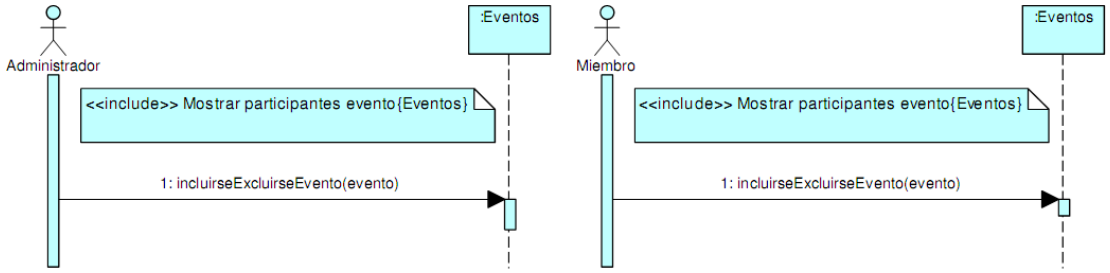
Participa como Administrador o Miembro del proyecto
evento.grupo.creador.usuarioRegistrado = self or evento.grupo.miembroAceptado.usuarioRegistrado -> includes(self)

Salida:
Muestra la lista de usuarios que participan en el evento

```
let participantesEvento: Set(usuarioRegistrado) = Set(evento.administradorParticipa.usuarioRegistrado)
                                                  -> union(evento.miembroParticipa.usuarioRegistrado)
```

Incluirse/Excluirse de evento

El miembro o administrador es incluido o excluido de la participación de un evento por deseo propio.



Operación: incluirseExcluirseEvento(evento: Evento)
Descripción: El usuario indica que quiere incluirse o excluirse de participar en el evento.
Precondiciones:
Debe existir el evento en el sistema
Evento.allInstances() -> exists(e | e = evento)

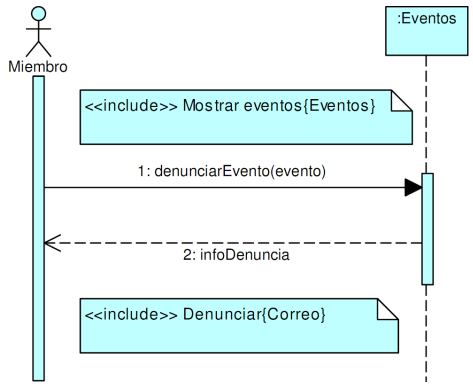
Participa como Administrador o Miembro del proyecto
evento.grupo.creador.usuarioRegistrado = self or evento.grupo.miembroAceptado.usuarioRegistrado -> includes(self)

Postcondiciones:
Si el miembro o administrador participaba en el evento queda excluído de este sino será incluido.

```
let infoUsuario: InfoUsuario = evento.grupo.infoUsuario -> select(i | i.usuarioRegistrado = self) in
if infoUsuario.membresía = TipoUsuario::Administrador then
  if evento.administradorParticipa -> excludes(infoUsuario) then
    evento.administradorParticipa -> includes(infoUsuario)
  else
    evento.administradorParticipa -> excludes(infoUsuario)
  endif
elseif infoUsuario.membresía = TipoUsuario::Miembro then
  if evento.miembroParticipa -> excludes(infoUsuario) then
    evento.miembroParticipa -> includes(infoUsuario)
  else
    evento.miembroParticipa -> excludes(infoUsuario)
  endif
endif
```

Denunciar evento

El miembro denuncia un evento indicando una razón.



Operación: denunciarEvento(evento: Evento, razón: String):
TupleType(razón: String, infoExtra: String)
Descripción: Obtiene los datos necesarios para formalizar la denuncia
Precondiciones:
Debe existir el evento en el sistema
Evento.allInstances() -> exists(e | e = evento)

Participa como miembro del proyecto al que pertenece el evento
evento.grupo.miembroAceptado.usuarioRegistrado -> includes(self)

Salida:
Se recopila la información necesaria para enviar.

```
let infoDenuncia : TupleType(razón: String, infoExtra: String) =
    Tuple{razón = razón,
        infoExtra = 'Tipo contenido: Evento, IdentificadorEvento: '
            -> concat(evento.identificador)
            -> concat(', IdentificadorGrupo: ')
            -> concat(evento.grupo.identificador)}
```

Apéndice III – Guía de utilización de Drupal

1. Interfaz y back-end

El núcleo de Drupal incorpora funcionalidades accesibles desde el back-end con una interfaz web de menús y formularios que se encuentran en la parte superior de la página, en la barra del administrador, accesible solo por el administrador del sistema.

Es posible crear diferentes tipos de contenido configurando los campos de los tipos básicos ya existentes. Cada contenido está basado en nodos. Un nodo es un trozo de contenido que es guardado en la BBDD y que son los que contienen la información de utilidad para el usuario que navega por la web.

Existe un apartado donde aparecen todas las estructuras que hay en el núcleo del Drupal como las páginas, bloques, nodos o tipos de contenido, menús, taxonomías. Además de otras instaladas por módulos adicionales como son los paneles o las vistas.

También existe un apartado donde son configurados los usuarios del sistema, sus roles y permisos.

Además es posible cambiar la apariencia del sitio web eligiendo entre los temas previamente descargados o creados.

El componente principal que proporciona las funcionalidades es el módulo, el cual también tiene un apartado para poder activarlos o desactivarlos además de instalarlos o desinstalarlos.

Incluye además un completo apartado para administrar el sistema que informa sobre los errores, mensajes de registro y acciones que se llevan a cabo por parte de los usuarios.

Y por último un menú para la configuración general del sistema Drupal. Tanto formatos de hora y fecha, idioma, página de inicio que se mostrará, además de muchas otras características. Una de las principales en el desarrollo web es la de mantenimiento la cual permite que el sitio no sea accesible por los usuarios salvo para el administrador del sistema para así poder realizar tareas de mantenimiento. La otra funcionalidad importante en el desarrollo es el vaciado de caché, el cuál es necesario cada vez que son realizados cambios en módulos.

El propósito de disponer de cachés en Drupal es por rapidez y eficiencia ya que registra en caché muchas de las partes de la página que no cambian, sobretodo para usuarios anónimos.

Bajo la barra de administrador se muestra la página web tal como la ven el resto de usuarios, siempre teniendo en cuenta los niveles de permisos, que son siempre diferentes para cada usuario.

2. Módulo

Los módulos son los componentes hechos en código PHP que sirven para extender las funcionalidades del sistema Drupal.

Cada módulo está compuesto de mínimo 3 archivos para su correcto funcionamiento. Uno de ellos es un archivo en formato texto con extensión .info. El cual proporciona información sobre el nombre del módulo en si, versión, versión Drupal con el cuál es compatible, además de listar el resto de archivos que pertenecen al módulo. También informa de las dependencias con otros módulos e incluso archivos css necesarios para la presentación.

Otro archivo no menos importante, también en formato PHP, con extensión .install. como se puede entrever, se encarga de la etapa de instalación y desinstalación del módulo. Por tanto, crea o elimina las tablas necesarias en la base de datos, además de configurar otros aspectos como variables de sistema.

El tercer y último archivo necesario para todo módulo, también en formato PHP, tiene extensión .module y es en éste donde reside toda la funcionalidad del módulo. En este archivo están todas las funciones que determinan como se comportará el módulo. Aunque por claridad es posible repartir ciertas funciones en otros archivos también en formato PHP y con extensión .inc.

3. Hooks

Entre todas las funcionas que componen los módulos, existen unas en particular llamadas hooks. Estas son funciones especiales que son ejecutadas en el momento adecuado dentro de los procesos de la web. Para conseguir esto son invocados en un orden y momento específicos que están determinados por el nombre de la propia función. La mayoría de estos hooks se nombran de la forma hook_(acción), en cada módulo que los implemente se debe cambiar la palabra hook por el nombre del módulo para así evitar colisión de nombres. Como ejemplo el hook *hook_node_insert* que se ejecuta justo después de haber insertado un nodo en la base de datos se debe nombrar para el módulo blog de la forma *blog_node_insert.*

Hay una API muy completa con todos los hooks que vienen en el núcleo de Drupal incluyendo tanto los necesarios para el funcionamiento del sistema como los opcionales.

Los hooks más importantes o utilizados son:

Para el archivo con extensión .install

hook_schema: Se encarga de definir las tablas que serán insertadas en la BBDD.
hook_install/hook_uninstall: Define las acciones a realizar al instalar o desinstalar un módulo excluyendo cambios en la BBDD, ya que de eso se encarga el hook anterior.

Para el archivos con extensión .module
hook_init: Actúa cada vez que es recibida una petición o query por parte de un navegador.
hook_menu: Define menús y rutas, asociándoles diversas características como que función será llamada para comprobar que las precondiciones son cumplidas y cual para realizar el procesado de datos.
hook_form: Define un formulario.
hook_form_validate: Valida un formulario.
hook_form_submit: Realiza el procesado después de validar el formulario para guardar los datos enviados a través del formulario web.
hook_block_info: Define un bloque.
hook_block_view: Retorna el código html que se mostrará dentro de un bloque.
hook_node_info: Define un nuevo tipo de nodo
hook_node_access: Indica si un usuario tiene permiso para realizar acciones sobre nodos.
hook_node_view: Modifica los datos a presentar justo antes de que el nodo sea visualizado.
hook_load/hook_node_load: Actúan cuando un nodo es cargado de la BBDD
hook_insert/hook_node_insert: Actúan cuando un nodo nuevo es creado.
hook_delete/hook_node_delete: Actúan cuando un nodo es eliminado.
hook_update/hook_node_update: Actúan cuando un nodo es actualizado.
hook_permission: Define permisos para roles de usuario.
hook_theme: Define las funciones o archivos que se encargan de generar código HTML.
hook_user_login: Es llamado cuando un usuario inicia sesión
hook_user_logout: Es llamado cuando un usuario cierra sesión.
hook_user_cancel/hook_user_delete: Actúan cuando la cuenta de un usuario es borrada

hook_user_insert: Actúa cuando un usuario ha sido creado.
hook_user_view: Actúa justo antes de que la cuenta de usuario vaya a ser visualizada.

Además, existen hook acabados en *alter* tales como *hook_schema_alter, hook_menu_alter, hook_form_alter*, *hook_user_view_alter, hook_node_view_alter,* etc. Que actúan justo después de sus funciones homólogas sin la terminación alter, para realizar cambios posteriores. Es útil cuando otro módulo ajeno al que implementa el hook sin terminación alter quiere realizar cambios de última hora sin alterar el flujo del primer módulo.

La cantidad de hooks enumerados es una pequeña parte de todos los que hay, además es posible crear hooks propios utilizando las funciones module_invoke y module_invoke_all nativas de Drupal. Ya que estas funciones llamarán a otras que estén declaradas con un formato determinado.

4. Nodos

Los nodos son pedazos de contenido principalmente compuestos de un título, un cuerpo y comentarios. Al crear un nuevo tipo de contenido pueden ser agregados nuevos campos. Además cada nodo puede ser publicado o reservado y también puede ser destacado en la página principal. Cada nodo además puede tener comentarios asociados y revisiones para tener constancia de quién y qué ha sido modificado y así poder deshacer los cambios si fuera necesario.

Es posible crear nuevos tipos a través de los ya existentes tanto a través del back-end del administrador como en código.

Cada campo de un nodo es guardado en la BBDD como una tabla individual para así poder ser aprovechado un mismo campo para varios tipos de nodo o contenido. Por tanto, a la hora de crear un nuevo tipo de contenido en código han de ser creados los campos necesarios y la instancias de estos, siendo estos últimos el vínculo entre un campo y el tipo de contenido.

Todo rol de usuario dispone de permisos configurables para la creación, edición y eliminación de nodos. Los cuáles existen para cada tipo de contenido ganando así granularidad a la hora de asignar los permisos.

Para la creación de tipos de contenido nuevos a través del código se utilizan los hooks, hook_install y hook_uninstall del archivo .install. Para definir el resto de características en los archivos .module se utilizan entre otros hook_node_info, para definir el nuevo nodo y hook_node_access para definir los permisos de los roles sobre el nodo. Además de los ya mencionados hook_node_view, hook_load/hook_node_load, hook_insert/hook_node_insert, hook_delete/hook_node_delete y hook_update/hook_node_update.

5. Campos

Los campos son las unidades básicas de las que se componen los nodos. Como ya se ha descrito anteriormente, cada campo puede estar asociado a más de un tipo de nodo/contenido.

A través del back-end de Drupal se pueden asociar nuevos campos para los tipos de nodo utilizando los tipos de campo definidos por los módulos. Los principales tipos de datos para los campos son Boolean, Decimal/Float, File, Image, Integer, List, Long text, Long text and summary, Term y Text. Aunque es posible crear nuevos tipos ya sea instalando nuevos módulos o creándolos en nuestros propios módulos.

Para la creación de campos existen varios hooks, los más usuales son, hook_field_info, que define un campo hook_field_schema, que define la tabla en la BBDD, hook_field_validate, que valida los datos introducidos por el usuario en el campo, entre otros muchos.

6. Base de datos

Toda la información generada en Drupal, además de sus estructuras y configuraciones son guardadas en la BBDD, la cual hasta la versión Drupal más reciente pueden ser mysql, postgresql o sqlite. Aunque es posible añadir más, siempre y cuando se disponga de los drivers necesarios.

La información sobre a que base de datos Drupal tiene que conectarse, el usuario, contraseña y configuración de la conexión está guardada en el fichero /sites/default/setting.php. del directorio principal de Drupal.
Drupal provee una capa de abstracción que nos permite desentendernos de la base de datos utilizada. Siendo muy útil si son manejadas varias. Como ejemplo para hacer una query no es necesario utilizar mysql_query() o pg_query(), simplemente db_query(). Esta función, entre otras, asegura además que las querys no contengan datos enviados por el usuario que puedan resultar en un ataque.

Hay un buen número de funciones y objetos útiles en esta capa de abstracción. Para realizar querys, se utiliza tanto db_query como db_select. En cambio db_select, db_insert, db_update y db_delete generan un objeto que se parametriza. Como ejemplo:

```
$num_updated = db_update('joke')
                    ->fields(array('title' => 'Qwerty',))
                    ->condition('nid', 3, '>=')
                    ->execute();
```

Existen un par de hooks relacionados con la BBDD que solo pueden ser emplazados en el archivo con extensión .install de los módulos, por tanto, cambios realizados en éstos solo serán visibles al instalarlos o desinstalarlos. Estos son hook_schema y hook_schema_alter, ya mencionados anteriormente.

7. Modelo Vista Controlador(MVC) en Drupal

Siendo el patrón por antonomasia en el desarrollo web, el cuál consiste en separar el modelo, la vista y el controlador en partes bien diferenciadas. El modelo está formado por todos los datos e información, la vista es el resultado de la conversión, filtrado y estructurado de esos datos de forma legible para el usuario a través del controlador que guarda la lógica del sistema y realiza los pasos necesarios para transformar los datos del modelo en datos de la vista.

Drupal como casi cualquier CMS utiliza este patrón, aunque no se ajusta del todo a este. Si que en rasgos generales, Drupal se apoya en una base de datos para guardar toda la información, además de otras configuraciones no mostrables al usuario habitual. Si que que dispone de vista, controlada por los temas y diferenciada de la lógica o controlador los cuáles serían los módulos.

La flexibilidad de Drupal nos permite que una pequeña parte de la vista esté dentro de los módulos, utilizando funciones theme en vez de plantillas. Y puede haber cierta lógica dentro de los temas, con las funciones process y preprocess. Por tanto, si que por lo general utiliza el patrón modelo-vista-controlador aunque no de forma estricta.

Apéndice IV – Glosario de términos

CMS *"(por sus siglas en inglés, Content Management System) es un programa que permite crear una estructura de soporte, framework, para la creación administración de contenidos, principalmente en páginas web, por parte de los administradores, editores, participantes y demás usuarios.*

Consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio web. El sistema permite manejar de manera independiente el contenido y el diseño." (Extracto de wikipedia)

Todo CMS aporta un marco de trabajo, proporcionando la gestión de usuarios y de contenido, seguridad y estructura principal de la página web. De esta manera posibilita no tener que empezar desde cero, además de dar herramientas para la fácil gestión de todo contenido.

HTML

Es el lenguaje utilizado en las web para la estructuración de la información. Compuesto por elementos en forma de etiquetas con atributos y que contienen la información. Está estructurado de forma que cada elemento puedo anidar cero o más elementos. Existe una organización, el W3C, que se encarga de armonizar las etiquetas y la forma en que los navegadores deben interpretarlas.

CSS

Es un lenguaje que describe a los navegadores como han de mostrar la información escrita en HTML y enviada desde el servidor. Determina posición y espaciado entre elementos, colores, fuentes, etc.

PHP

"PHP es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web que genera la página Web resultante. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo."(Extracto de wikipedia)

Este lenguaje es el más ampliamente utilizado por las páginas web. Genera contenido dinámico, posibilitando que este contenido cambie según diversos factores, tales como el usuario lector, la incorporación de nuevo contenido de forma continua e información externa relacionada y que influya en ese contenido.

Crowdfunding

Consiste en la financiación colectiva de proyectos, campañas o como ayuda financiera a través de pequeñas donaciones de particulares.